Instituto Politécnico de Tomar

**Escola(s) Superior de Tecnologia de Tomar**

**Ricardo Santos Anacleto**

# Comparação do desempenho de diferentes abordagens para o desenvolvimento de aplicações móveis

Dissertação de Mestrado

Orientado por:

Luís Oliveira – IPT

Renato Panda - IPT

Dissertação apresentada ao Instituto Politécnico de Tomar
para cumprimento dos requisitos necessários
à obtenção do grau de Mestre em
Engenharia Informática – Internet das Coisas

# Resumo

Esta dissertação teve como objectivo principal a comparação de diversas tecnologias existentes para o desenvolvimento de aplicações móveis e componentes associados. Para o efeito foram definidos diversos testes com métricas e objectivos concretos. O resultado destes testes deu origem a um conjunto de recomendações que servem de base tanto ao projecto associado (VitaSenior) como a futuros projectos de software.

Tendo sempre como principal foco a satisfação das necessidades do respectivo projecto, foram delineados alguns testes a realizar para então após os testes se conseguir fazer uma escolha assertiva da tecnologia a desenvolver a presença mobile do mesmo.

Os testes seleccionados e respectivas finalidades foram então os seguintes:

- Medição do tempo em milissegundos da execução de pedidos para as rotas de login e registo da interface de programação de aplicações (do inglês Aplication Programing Interface – API) com a finalidade de medir os tempos médios necessários à realização de uma operação de rotina da aplicação.
- Medição do tempo em milissegundos da execução de uma tarefa de uso intensivo do processador, quer numa única thread como em múltiplos threads, com a finalidade de verificar diferenças de desempenho caso seja necessária efectuar tal operação.
- Medição do tempo em milissegundos do carregamento de imagens com endereçamento local e remoto, de modo a determinar a detentora da melhor performance.
- Medição do tempo de execução em milissegundos de uma série de pedidos encadeados de forma a verificar a performance média de execução de pedidos à API.
- Medição do consumo de memória e verificação do mecanismo de gestão de memória.
- Medição do consumo de bateria máximo estimado.

Sendo também efectuada uma análise da simplicidade e facilidade de manutenção de código bem como uma comparação da usabilidade entre nativo e uma outra tecnologia/framework abordada.

Relativamente aos pedidos de rede, foi também avaliada a capacidade de resposta da API a utilizar no projecto do VITASENIOR, onde foram efectuados todos os pedidos de rede realizados, sendo realizados testes de carga com a aplicação JMeter, determinando então o número de pedidos médio a que a interface de programação de aplicações (do inglês Aplication Programing Interface – API) e diversas rotas utilizadas é capaz de dar resposta sem interferir na performance da aplicação.

Para a realização das diversas aplicações de teste foram seleccionadas após um estudo inicial das tecnologias/frameworks existentes, Android nativo, aplicação web progressiva (do inglês Progressive Web App - PWA) com interface gráfica desenvolvida com a framework VueJS e desenvolvimento recorrendo a uma tecnologia de desenvolvimento hibrido neste caso IONIC, com código elaborado com recurso a AngularJS.

Após a elaboração de todos os testes e análises verificou-se que Android nativo apresenta uma melhor performance em termos gerais, sendo que a PWA apresenta um melhor equilíbrio entre performance, simplicidade de código, facilidade de manutenção e permite obter presença em todas as plataformas com apenas uma base de código desenvolvida.

Verificou-se ainda que a usabilidade nativa de Android pode ser replicada na PWA, sendo também o consumo de bateria máximo expectável equivalente. Constatou-se contudo que o browser onde a PWA corre terá também impacto na sua performance, sendo no entanto esta a escolha para um cenário onde o pretendido é um equilíbrio entre todos os factores mencionados e testados.

**Palavras-chave (Tema):** **Dispositivos móveis, testes não funcionais**.

**Palavras-chave (Tecnologias):** **JMeter, Android, PWA, IONIC, VueJS, AngularJS.**

# Agradecimentos

Gostaria de agradecer a todos os que me ajudaram não só no decorrer desta dissertação, toda a minha família e a minha mulher por a compreensão do tempo que tive de estar um pouco mais “afastado” e as férias adiadas, como também aos meus amigos e colegas André Farinha, Pedro Ferreira e Nelson Gomes por todo o trabalho que realizamos juntos sempre com espirito de equipa e entreajuda.

Gostaria também de agradecer aos meus orientadores Luís Oliveira e Renato Panda por tempo disponibilizado e ajuda prestada na realização desta dissertação.

E também ao meu filhote Diogo Anacleto por não ter nascido antes e deixado o pai terminar a dissertação ☺ com esperança que um dia o possa ajudar na dele e até lá em todo o resto.

# Índice

*Resumo* ... *v*

*Agradecimentos* ... *vii*

*Índice* ... *ix*

*Índice de Figuras* ... *xiii*

*Índice de Tabelas* ... *xvii*

*Notação e Glossário* ... *xix*

*1 Introdução* ... *1*

*2 Estudo prévio* ... *3*

**2.1 Identificação de tecnologias** ... **3**

2.1.1 Desenvolvimento nativo ... 3

2.1.2 Desenvolvimento multiplataforma ... 6

**2.2 Testes** ... **15**

2.2.1 Testes funcionais e não funcionais ... 16

2.2.2 Principais tipos de testes não funcionais ... 17

2.2.2.1 Testes de performance ... 17

2.2.2.2 Testes de carga ... 17

2.2.2.3 Testes de stress ... 17

2.2.2.4 Testes de usabilidade ... 17

2.2.2.5 Manutenção ... 18

2.2.2.6 Fiabilidade ... 18

2.2.2.7 Portabilidade ... 18

2.2.3 Testes dinâmicos e testes estáticos ... 18

**2.3 Escolha dos tipos de testes** ... **19**

*3 Solução proposta* ... *21*

**3.1 Identificação de pontos a testar** ... **21**

**3.2 Selecção de tecnologias ... 22**

**3.3 Hardware e largura de banda ... 23**

**3.4 Testes à API ... 24**

3.4.1 Quantos pedidos o hardware de teste é capaz de fazer? ... 25

3.4.2 Quantos pedidos por segundo a instancia é capaz de dar resposta? ... 26

3.4.3 Quantos pedidos por segundo a API é capaz de dar resposta? ... 26

3.4.4 A capacidade de resposta da API é igual para todos as rotas? ... 28

**3.5 Aplicações de Teste ... 29**

3.5.1 Medição de tempos de acesso para Login / Registo ... 30

3.5.1.1 Android ... 30

3.5.1.2 PWA ... 33

3.5.1.3 IONIC ... 36

3.5.2 Carregamento do armazenamento local ... 38

3.5.2.1 Android ... 39

3.5.2.2 PWA ... 39

3.5.2.3 IONIC ... 41

3.5.3 Carregamento por endereço da internet ... 42

3.5.3.1 Android ... 42

3.5.3.2 PWA ... 43

3.5.3.3 IONIC ... 46

3.5.4 Verificação de tempos de execução de tarefa intensiva de processador em Single Thread ... 47

3.5.5 Verificação de tempos de execução de tarefa intensiva de processador em Multi Thread ... 50

3.5.5.1 Android ... 50

3.5.5.2 PWA ... 54

3.5.5.3 IONIC ... 56

3.5.6 Operações de rede ... 57

3.5.6.1 Android ... 57

3.5.6.2 PWA ... 58

3.5.6.3 IONIC ... 60

*4 Gestão de memória ........ 63*

**4.1 Android ........ 63**

**4.2 PWA ........ 63**

**4.3 IONIC ........ 64**

*5 Usabilidade ........ 65*

*6 Consumo de bateria máximo estimado ........ 67*

**6.1 Android ........ 67**

**6.2 PWA ........ 67**

**6.3 IONIC ........ 68**

*7 Simplicidade de código e facilidade de manutenção ........ 69*

**7.1 Simplicidade do código ........ 69**

**7.2 Facilidade de manutenção do código ........ 74**

*8 Resultados ........ 75*

*9 Conclusões ........ 81*

**9.1 Limitações & trabalho futuro ........ 82**

*10 Bibliografia ........ 83*

# Índice de Figuras

*Figura 1 – Arquitectura Android* ........ 4

*Figura 2 - Preços praticados nas stores* ........ 5

*Figura 3 – Arquitectura IONIC* ........ 7

*Figura 4 – Arquitectura Cordova* ........ 8

*Figura 5 - Comparação de suporte JIT/AOT/x64* ........ 8

*Figura 6 - Comparação de suporte hot reload/hot swap* ........ 9

*Figura 7 – Funcionamento do Service Worker* ........ 10

*Figura 8 – Arquitectura PWA* ........ 11

*Figura 9 - Demonstração do termo APP SHELL* ........ 12

*Figura 10 - Tabela de compatibilidade do service worker* ........ 13

*Figura 11 - Comparação de frameworks de desenvolvimento Mobile* ........ 14

*Figura 12 - BenchMark de diversas frameworks JavaScript* ........ 15

*Figura 13 - Testes funcionais Vs. não funcionais* ........ 16

*Figura 14 – Teste realizado em : http://speedmeter.fccn.pt/* ........ 24

*Figura 15 – Teste realizado em: http://www.speedtest.net/pt* ........ 24

*Figura 16 – Máximo de pedidos registado sem conexão de rede activa* ........ 25

*Figura 17 – Média de pedidos por segundo registados* ........ 25

*Figura 18 – Máximo de pedidos à rota desconhecida registados* ........ 26

*Figura 19 – Utilização de ficheiro CSV com campos necessários ao pedido* ........ 27

*Figura 20 – Utilização de parâmetros mapeados do ficheiro CSV* ........ 27

*Figura 21 – Pedidos por segundo para rota registar* ........ 28

*Figura 22 – Pedidos por segundo médios registado nas operações de login* ........ 28

*Figura 23 – Pedidos por segundo médios registados nas operações de Listagem de VITABOX's* ........ 28

*Figura 24 – Pedidos por segundo médios registados nas operações de listagem de utilizadores de uma box.* ........ 28

*Figura 25 – Representação gráfica dos resultados obtidos para rota login (Android)* ........ 31

*Figura 26 - Representação gráfica dos resultados obtidos para rota registar (Android)* ........ 32

*Figura 27 – Representação gráfica dos resultados obtidos para rota login (PWA)......................................33*

*Figura 28 - Representação gráfica dos resultados obtidos para rota login (PWA Instalada) ........................34*

*Figura 29 - Representação gráfica dos resultados obtidos para rota registo (PWA) ....................................35*

*Figura 30 - Representação gráfica dos resultados obtidos para rota registo (PWA instalada)......................36*

*Figura 31 - Representação gráfica dos resultados obtidos para rota login (IONIC)....................................37*

*Figura 32 - Representação gráfica dos resultados obtidos para rota registar (IONIC) ................................38*

*Figura 33 – Representação gráfica dos resultados obtidos no carregamento de 100 imagens de armazenamento interno (Android)........................................................................................................39*

*Figura 34 – Representação gráfica dos resultados obtidos no carregamento de 100 imagens do armazenamento interno (PWA)............................................................................................................40*

*Figura 35 - Representação gráfica dos resultados obtidos no carregamento de 100 imagens do armazenamento interno (PWA instalada)...........................................................................................41*

*Figura 36 - Representação gráfica dos resultados obtidos no carregamento de 100 imagens de armazenamento interno (IONIC) ..........................................................................................................42*

*Figura 37 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (Android)......................................................................................................................................43*

*Figura 38 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (PWA)...........................................................................................................................................44*

*Figura 39 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (PWA instalada)...........................................................................................................................45*

*Figura 40 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (IONIC).........................................................................................................................................46*

*Figura 41 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (Android)...........................................................................................................................................47*

*Figura 42 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (PWA)..................................................................................................................................................48*

*Figura 43 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (PWA instalada).................................................................................................................................49*

*Figura 44 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (IONIC).................................................................................................................................................50*

*Figura 45 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (Android Main Thead) ..........................................................................................................51*

*Figura 46 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (Android AsyncTask)* ........ 52

*Figura 47 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (Android RxJava)* ........ 53

*Figura 48 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (PWA)* ........ 54

*Figura 49 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (PWA instalada)* ........ 55

*Figura 50 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (IONIC)* ........ 56

*Figura 51 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (Android)* ........ 58

*Figura 52 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (PWA)* ........ 59

*Figura 53 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (PWA instalada)* ........ 60

*Figura 54 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (IONIC)* ........ 61

*Figura 55 – Gestão de memória Android* ........ 63

*Figura 56 – Gestão de memória PWA* ........ 64

*Figura 57 – Gestão de memória IONIC* ........ 64

*Figura 58 – Interligação de peidos de rede PWA* ........ 70

*Figura 59 – Interligação de pedidos IONIC* ........ 70

*Figura 60 – Interligação de pedidos Android* ........ 71

*Figura 61 – Exemplo de código tarefa de utilização intensiva do processador (PWA)* ........ 72

*Figura 62 - Exemplo de código tarefa de utilização intensiva do processador (IONIC)* ........ 73

*Figura 63 - Exemplo de código tarefa de utilização intensiva do processador (Android)* ........ 73

# Índice de Tabelas

*Tabela 1 – Tempos registados para rota login (Android)* 31

*Tabela 2 - Tempos registados para rota registar (Android)* 32

*Tabela 3 – Tempos registados para rota login (PWA)* 33

*Tabela 4 - Tempos registados para rota login (PWA Instalada)* 34

*Tabela 5 - Tempos registados para rota registar (PWA)* 35

*Tabela 6 - Tempos registados para rota registar (PWA Instalada)* 35

*Tabela 7 - Tempos registados para rota login (IONIC)* 36

*Tabela 8 - Tempos registados para rota registar (IONIC)* 37

*Tabela 9- Tempos de carregamento de 100 imagens de armazenamento interno (Android)* 39

*Tabela 10 – Tempos de carregamento de 100 imagens de armazenamento interno (PWA)* 40

*Tabela 11 - Tempos de carregamento de 100 imagens de armazenamento interno (PWA instalada)* 41

*Tabela 12 - Tempos de carregamento de 100 imagens de armazenamento interno IONIC* 42

*Tabela 13 - Tempos de carregamento de 25 imagens de endereço da internet (Android)* 43

*Tabela 14 - Tempos de carregamento de 25 imagens de endereço da internet (PWA)* 44

*Tabela 15 - - Tempos de carregamento de 25 imagens de endereço da internet (PWA instalada)* 45

*Tabela 16 - Tempos de carregamento de 25 imagens de endereço da internet (IONIC)* 46

*Tabela 17 - Tempo de execução de tarefa intensiva em single thread (Android)* 47

*Tabela 18 - Tempo de execução de tarefa intensiva em single thread (PWA)* 48

*Tabela 19 - Tempo de execução de tarefa intensiva em single thread (PWA instalada)* 49

*Tabela 20 - Tempo de execução de tarefa intensiva em single thread (IONIC)* 49

*Tabela 21 - Tempo de execução de 32 tarefas intensiva em multi thread (Android Main Thread)* 51

*Tabela 22 - Tempo de execução de 32 tarefas intensiva em multi thread (Android AsyncTask)* 52

*Tabela 23 - Tempo de execução de 32 tarefas intensiva em multi thread (Android RxJava)* 53

*Tabela 24- Tempo de execução de 32 tarefas intensiva em multi thread (PWA)* 54

*Tabela 25 - Tempo de execução de 32 tarefas intensiva em multi thread (PWA instalada)* 55

*Tabela 26 - Tempo de execução de 32 tarefas intensiva em multi thread (IONIC)* 56

*Tabela 27 – Registo de tempos de execução de pedidos de rede interligados (Android)* 57

*Tabela 28 - Registo de tempos de execução de pedidos de rede interligados (PWA)* ____________________58

*Tabela 29 - Registo de tempos de execução de pedidos de rede interligados (PWA instalada)* ___________59

*Tabela 30 - Registo de tempos de execução de pedidos de rede interligados (IONIC)* _________________60

*Tabela 31 – Comparação geral dos tempos obtidos* ______________________________________________75

*Tabela 32 – Comparativo global dos resultados das PWA's* ______________________________________76

*Tabela 33 – Comparativo de resultados PWA nos browsers Chrome e Opera* _______________________76

*Tabela 34 – Dois melhores resultados dos testes das PWA* ______________________________________77

*Tabela 35 – Comparação final "filtrada" entre tecnologias/frameworks* ____________________________77

*Tabela 36 – Exemplo de sugestão de tecnologia/Framework a utilizar* ______________________________78

# Notação e Glossário

| | |
|---|---|
| **API** | Aplication Programing Interface |
| **DDOS** | Distributed Denial of Service |
| **DOM** | Document Object Model |
| **DOS** | Denial Of Service |
| **ISTQB** | International Software Testing Qualifications Board |
| **ms** | Milissegundos |
| **PWA** | Progressive Web App |

# 1 Introdução

No decorrer da elaboração do projecto VITASENIOR, um projecto que teve origem na actual realidade do nosso país estar cada vez mais envelhecido e a necessidade de um acompanhamento a idosos cada vez mais isolados e necessidade da selecção de uma tecnologia ou framework apropriada para presença móvel.

Sendo que actualmente existem inúmeras formas de criar aplicações móveis, desde forma nativa a frameworks de desenvolvimento multiplataforma, e uma lacuna na comparação efectiva de pontos concretos referentes às suas performances, surgiu a necessidade de identificar de forma objectiva com recurso a testes com métricas definidas. Para satisfação dessa necessidade será efectuada uma selecção de uma amostra entre as tecnologias/ frameworks abordadas num estudo prévio á realização dos testes.

Os testes que se resolveu efectuar foram determinados com base nas necessidades básicas do projecto, nomeadamente a realização de pedidos à API, visualização de dados e imagens e rapidez de execução das tarefas, tendo sido definidos os seguintes testes e suas respectivas finalidades:

- Medição do tempo em milissegundos da execução de pedidos para as rotas de login e registo da interface de programação de aplicações (do inglês Aplication Programing Interface – API) com a finalidade de medir os tempos médios necessários à realização de uma operação de rotina da aplicação.
- Medição do tempo em milissegundos da execução de uma tarefa de uso intensivo do processador, quer numa única thread como em múltiplos threads, com a finalidade de verificar diferenças de desempenho caso seja necessária efectuar tal operação.
- Medição do tempo em milissegundos do carregamento de imagens com endereçamento local e remoto, de modo a determinar a detentora da melhor performance.
- Medição do tempo de execução em milissegundos de uma série de pedidos encadeados de forma a verificar a performance média de execução de pedidos à API.
- Medição do consumo de memória e verificação do mecanismo de gestão de memória.
- Medição do consumo de bateria máximo estimado.

Sendo também efectuada uma análise da simplicidade e facilidade de manutenção de código bem como uma comparação da usabilidade entre nativo e uma outra tecnologia/framework abordada.

A realização destes testes e verificações tem como objectivo final não só a satisfação da necessidade mencionada, mas que com os resultados se possa de forma genérica orientar com factos, a correcta escolha da tecnologia a utilizar num projecto com necessidade de presença mobile.

# 2 Estudo prévio

Neste capítulo ou secção é feita uma revisão das diversas soluções encontradas, sendo efectuada uma breve análise dos pontos fortes e dos pontos fracos das mesmas, bem como um estudo dos tipos de testes de performance a realizar tanto às aplicações como à API que irá ser utilizada. Sendo esta secção subdividida nesses mesmos dois pontos identificados, enumeração das tecnologias e testes possíveis realizar tanto às aplicações como à API.

## 2.1 Identificação de tecnologias

Sendo o âmbito uma aplicação móvel iniciou-se o estudo na componente nativa, sendo efectuado apenas para as principais plataformas existentes no mercado, Android e IOS, não sendo contemplados outros sistemas operativos devido a sua fraca ou quase inexistente quota de mercado, do mesmo modo foi excluída a plataforma Windows Phone por o mesmo motivo e ter sido oficialmente descontinuada.

### 2.1.1 Desenvolvimento nativo

Dentro do desenvolvimento nativo Android, temos disponível o desenvolvimento em Java e Kotlin, sendo esta ultima a linguagem e o "caminho" cada vez mais recomendado por a Google, corrigindo alguns pontos do Java, mas não dispondo pelo menos por enquanto de algumas funcionalidades que o Java dispõem, sendo estas listadas e podendo ser analisadas em mais detalhe no site [1], entre essas funcionalidades encontra-se por exemplo verificação de valores nulos ou mesmo delegação, entre outros pontos descritos no site anteriormente referido.

Esta será a forma de execução mais próxima da camada física do hardware sendo a camada mais baixa a Kernel de linux onde se pode encontrar todos os drivers referentes ao hardware do dispositivo, numa camada superior encontramos as livrarias incluindo as livrarias base que em conjunto com a maquina virtual onde o sistema corre compõem o runtime do Android, acima desta camada existe ainda uma outra onde encontramos a framework aplicacional onde as aplicações podem aceder de forma a utilizar os recursos do sistema, podendo verificar esta estrutura na retirada de [2].

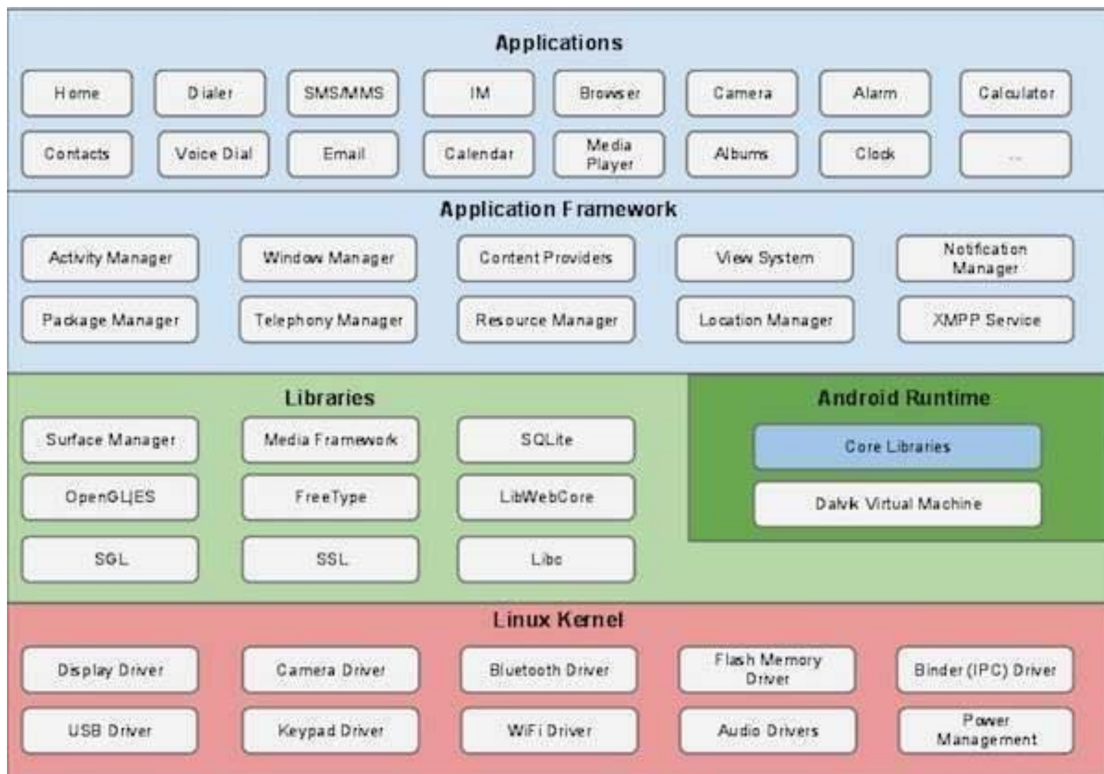



*Figura 1 – Arquitectura Android*

Com desenvolvimento para esta plataforma irá ser possível obter presença na Google Play Store, estando directamente ligada à maioria ou quase totalidade dos utilizadores com telemóvel que corra a mesma plataforma, dando uma boa visibilidade à aplicação, sendo que é necessário um pagamento único que ronda os 25€ no presente momento.

Como plataforma nativa e especificamente desenvolvida ou ajustada pelo fabricante para uso no dispositivo, a performance das aplicações será determinada pelo hardware e pelo desempenho do código realizado.

Sendo mantido pela Google, mas um projecto de Open Soure, tem a garantia de continuidade e continuo desenvolvimento de novas funcionalidades bem como correcção de eventuais bugs e vulnerabilidades, bem como a contribuição da comunidade.

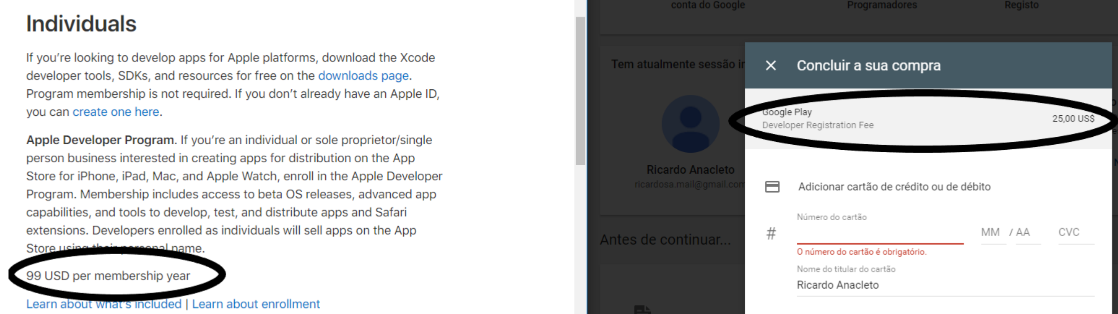



*Figura 2 - Preços praticados nas stores*

Ainda relativamente ao desenvolvimento nativo, existe também o IOS, plataforma da Apple que dispõem igualmente também de duas linguagens para desenvolvimento, Objective-C e SWIFT, ambas linguagens desenvolvidas pela marca, SWIFT veio tornar o desenvolvimento para mac-os e iOS mais ágil, simples e produtivo. Dotada de uma maior facilidade de utilização bem como aprendizagem e consequente aumento de produtividade, Objective-C torna-se mais antiquado e escolha geralmente apenas para projectos já desenvolvidos nessa linguagem, existindo mesmo assim a possibilidade de desenvolver com swift em projectos de Objective-C. Sendo estas e outras características enumeradas em estudos como o presente em [3].

Existindo igualmente uma plataforma de distribuição embora com uma quota de mercado menor que o Android, existem também milhões de utilizadores, tem como contra o pagamento que ronda os 99 dólares americanos á presente data, sendo estes de pagamento anual e não único como a plataforma concorrente. Plataforma também ela desenvolvida esta, especificamente para o hardware produzido, é expectável uma performance eximia.

Algo que pode ser considerado como uma outra desvantagem é o facto que para efectuar desenvolvimento para esta plataforma é necessário possuir um computador da marca com acesso à sua AppStore de forma a efectuar o download de um programa especifico (xcode) onde o desenvolvimento é realizado, sendo este também o único editor disponível para o desenvolvimento de código nativo para esta mesma plataforma.

### 2.1.2 Desenvolvimento multiplataforma

Relativamente ao desenvolvimento multiplataforma, existe tipicamente as frameworks denominadas de híbridas, sendo que estas possibilitam o desenvolvimento numa única linguagem, mas que com maiores ou menores adaptações, esse código tem a possibilidade de ser exportado tanto para web como para diversas plataformas nativas identificadas em cada uma das frameworks. No entanto foi também identificada uma relativamente nova tecnologia que e encontra a ser cada vez mais adoptada verificando-se o seu crescimento e adopção, as Progressive Web App - PWA, sendo por tal facto também aqui abordada como alternativa.

Quanto ao desenvolvimento híbrido foram analisadas três soluções que são as mais vulgarmente utilizadas no desenvolvimento deste tipo de projectos, Xamarim, React Native e Ionic. Cada uma destas soluções apresenta vantagens de desvantagens, tentando-se enumerar algumas das suas vantagens e desvantagens tentando efectuar uma análise comparativa das mesmas para o futuro auxílio à escolha da tecnologia a utilizar na realização de uma das aplicações de teste.

A arquitectura das aplicações deste tipo são normalmente um pouco mais complexas não sendo o código desenvolvido corrido de forma directa como as nativas, fazendo recurso a uma webview que corre então dentro da aplicação que irá correr nativamente, alem deste facto a forma de funcionamento e como a interligação de código é gerada depende da framework que é utilizada, neste caso damos como exemplo a arquitectura da framework IONIC demonstrada na Figura 3 retirada de [4], bem como a arquitectura de Cordova, normalmente utilizada para exportação de código multiplataforma demonstrada na Figura 4 retirada de [5].

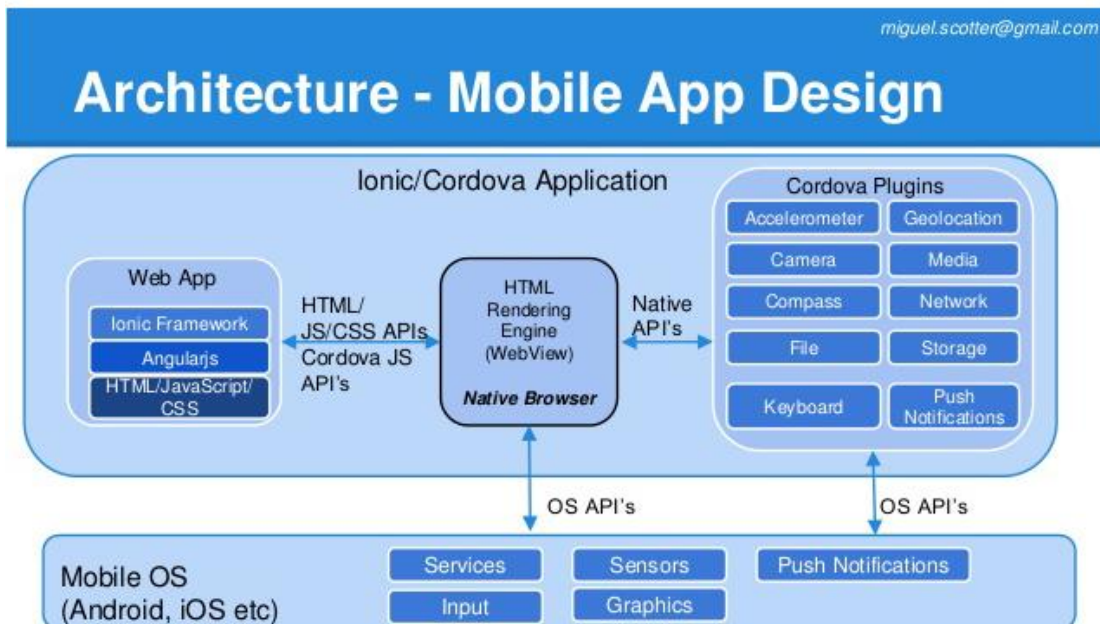



*Figura 3 – Arquitectura IONIC*

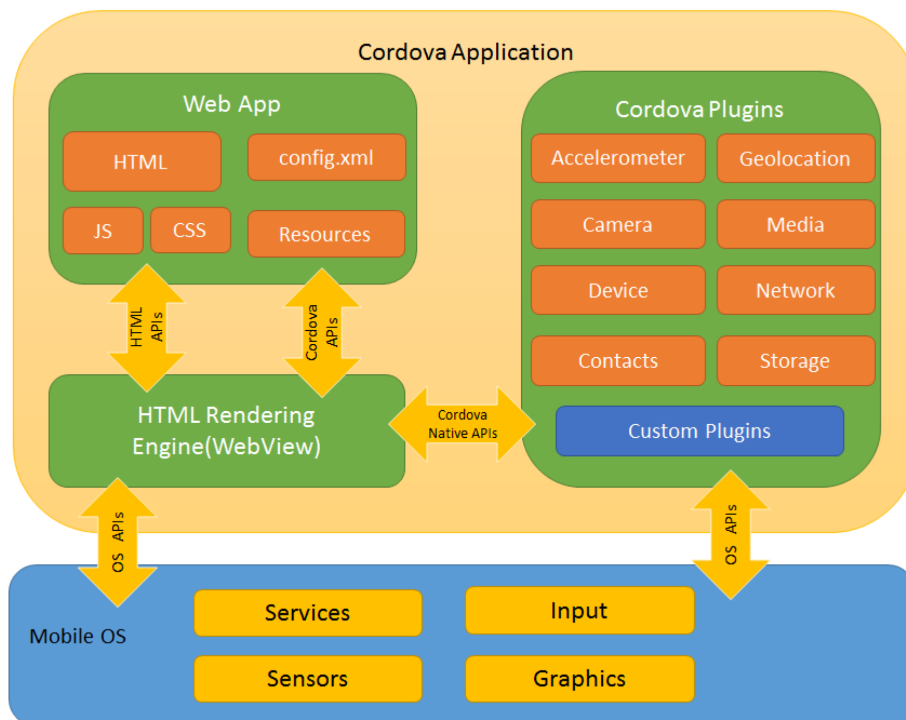



*Figura 4 – Arquitectura Cordova*

Iniciando pelo método de como as mesmas geram as interfaces, tanto Xamarim como React Native geram código nativo sendo que Ionic gera meramente html que irá ser corrido numa webview nativa a cada plataforma.

Relativamente a performance e formas de compilação podemos verificar que existem algumas diferenças entre cada uma das frameworks sendo JIT, AOT e compatibilidade com 64 bits, suportado independentemente por algumas das frameworks e não por outras como pode ser verificado na Figura 5 retirada de [6].

| | | Xamarin | React Native | Ionic |
|---|---|---|---|---|
| Running the code | iOS | AOT | Interpreter | Interpreter JIT with plugin |
| | Android | JIT/AOT | JIT | JIT |
| 64-bit support | iOS | YES | ? | Yes |
| | Android | Yes | No | ? |
| GUI | iOS and Android | Native widgets (directly or under the hood) | Native widgets under the hood | HTML |

*Figura 5 - Comparação de suporte JIT/AOT/x64*

Ainda no âmbito da performance, mas desta vez relativa à produtividade no desenvolvimento de código podemos também verificar que não existe um suporte transversal da capacidade de efectuar deploy das alterações ao código rapidamente (hot reload / code swap) que permite ao developer verificar se a sua alteração obteve resultados desejados ou não de forma rápida e eficaz, sendo a cobertura desta funcionalidade facilmente verificada na Figura 6 retirada de [6].

| | | Official tools | Xamarin | React Native | Ionic |
|---|---|---|---|---|---|
| Manual restarting | | Yes | Yes | Yes | Yes |
| Automatic restarting | | No | No | Live/Hot reloading | Yes |
| Hot swapping code | iOS | No | No | Hot reloading | No |
| | Android | Instant run | No | Hot reloading | No |
| Cold swapping code | iOS | No | No | Live reloading | Yes |
| | Android | Instant run | Fast deployment | Live reloading | Yes |
| Development in browser | | No | No | No | Looks differently |
| Instant updates | iOS | No | No | Yes | Yes |

*Figura 6 - Comparação de suporte hot reload/hot swap*

Uma outra desvantagem comum a todo o tipo de desenvolvimento híbrido é o facto de inevitavelmente cada uma das plataformas ter as suas próprias especificidades tal como o sistema de permissões, que é bastante diferente em ambas as plataformas e cada uma tem as suas próprias permissões que poderão ser pedidas, embora o código especifico para cada plataforma seja sempre muitíssimo mais reduzido que a realização de código nativo.

Passando ao conceito de PWA surgiu relativamente a dois anos, projecto também de código aberto da Google, demarca-se com algumas normas para a sua implementação, permitindo resultados bastante bons a nível de desempenho. Para além do desempenho é também detentor de algumas características como a obrigatoriedade da utilização de HTTPS, estando esta relacionada com o service worker, este é uma funcionalidade que nos permite aceder a conteúdo em cache, podendo ser o conteúdo consultado mesmo que o dispositivo não possua actualmente ligação à Internet, actuando este como um proxy como demonstrado na Figura 7 retira de [7].

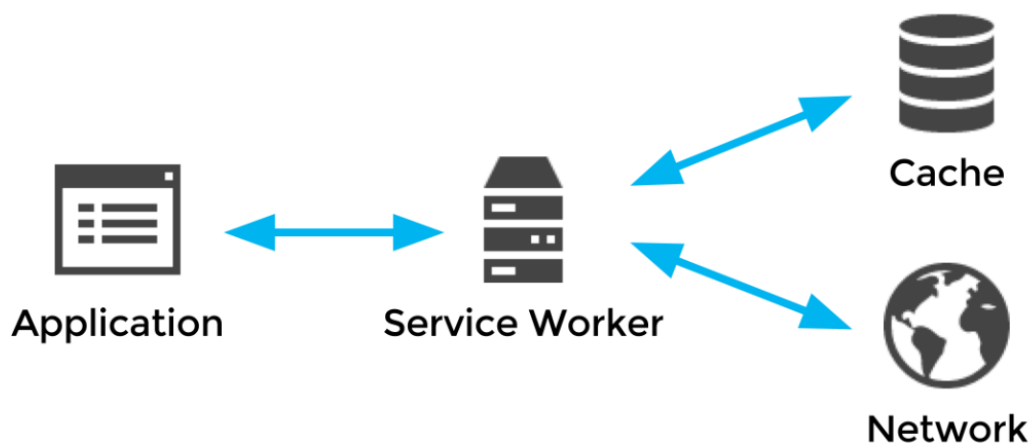



*Figura 7 – Funcionamento do Service Worker*

Apresentando também uma arquitectura um pouco mais simples, devido ao facto de ser na sua base um website, esta irá correr directamente no browser como outro qualquer tipo de site, recorrendo às apis disponibilizadas nesse mesmo browser, podendo-se verificar a sua arquitectura na Figura 8 retirada de [8].

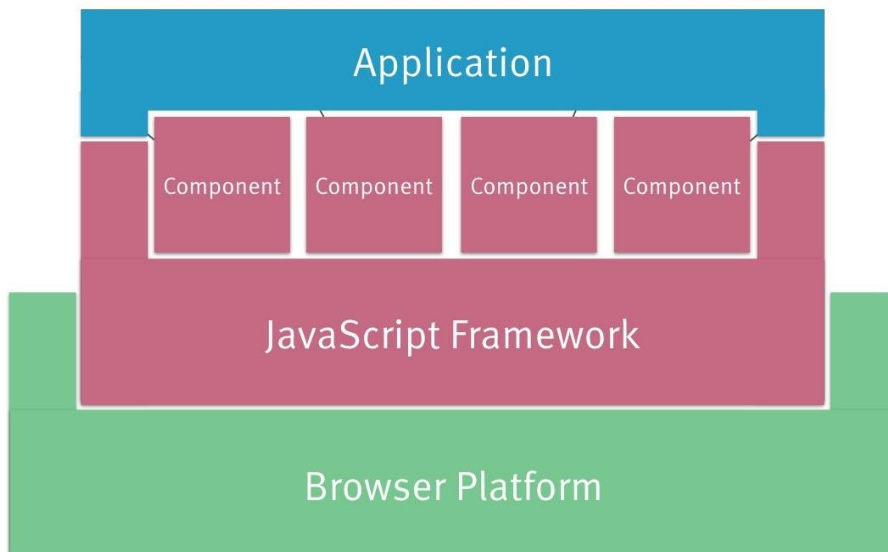



*Figura 8 – Arquitectura PWA*

Tendo também a vantagem de se poder utilizar frameworks para o desenvolvimento do front end, permitindo assim aliar novas funcionalidades à produtividade oferecida pelas mesmas, ou simplesmente a comodidade de desenvolver numa linguagem já conhecida.

Recorrendo aos padrões mencionados no site do projecto onde poderão ser consultadas mais informações [9], com recurso a utilização e carregamento inicial de uma interface essencial sem dados denominada de shell, apenas conteúdo básico para o utilizador iniciar o seu contacto, indo progressivamente após este processo estar terminado, carregando o seu conteúdo como se pode verificar na Figura 9 retirada de [10], é possível iniciar o site rapidamente e manter o utilizador activo no site, sendo este aspecto importante devido ao facto de num estudo efectuado se ter verificado que o utilizador em média abandona o site caso este não seja carregado em 3 segundos.

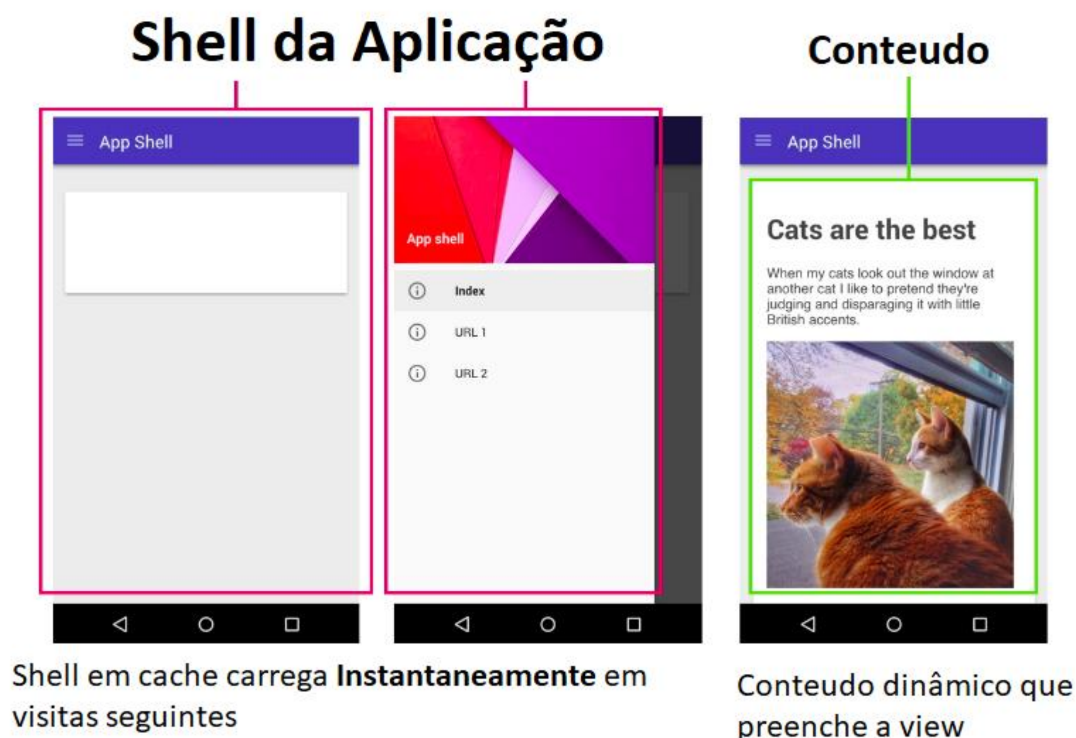



*Figura 9 - Demonstração do termo APP SHELL*

Para alem destas funcionalidades tem também presente a característica de se "auto instalar" no dispositivo mediante uma permissão concedida pelo utilizador, permissão esta que é pedida ao mesmo de acordo com alguns critérios:

- Possuir um ficheiro manifest.json;
- Ter Service Worker implementado;
- Ter HTTPS implementado;
- Ser a segunda visita ao site com um espaçamento mínimo de 5 minutos.

Tendo também como possibilidade a utilização de vários recursos nativos dos dispositivos através de API disponibilizadas pelos browsers, estando a ser actualizados constantemente.

O seu progresso pode ser visualizado no site what web can do today [11], sendo que uma componente em falta importante é a actual falta de implementação do Service Worker por parte da Apple no seu browser (safari), sendo que já se encontra em desenvolvimento no

safari tecnoligy preview. A sua compatibilidade pode ser verificada na Figura 10 retirada de [12].

Compatibilidade dos browsers

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Basic support | 40 | 17 | 44 * | No | 27 | 11.1 | 40 | 40 | ? | 44 | 27 | 11.1 | 4.0 |
| scriptURL | 40 | 17 | 44 * | No | 27 | 11.1 | 40 | 40 | ? | 44 | 27 | 11.1 | 4.0 |
| state | 40 | 17 | 44 * | No | 27 | 11.1 | 40 | 40 | ? | 44 | 27 | 11.1 | 4.0 |
| onstatechange | 40 | 17 | 44 * | No | 27 | 11.1 | 40 | 40 | ? | 44 | 27 | 11.1 | 4.0 |

Suporte completo
Sem suporte
Compatibilidade desconhecida
Experimental, expectavel alteração do comportamento.
Verificar notas de implementação
Utilizador tem expressamente de activar a funcionalidade

*Figura 10 - Tabela de compatibilidade do service worker*

Existindo para esta opção a utilização de frameworks, foram também analisadas a frameworks mais comuns de desenvolvimento de front ends JavaScript podendo verificar a sua popularidade na Figura 11 retirada de [13], sendo estas, React, AngularJs e VueJs, seguindo-se a análise das mesmas seguidamente.

ReactJS é framework recente, mas com o um o facto bastante positivo de ser desenvolvida e mantida pelo Facebook. Dotada de um bom desempenho, renderizando a pagina pretendida rapidamente, devido ao facto de que utiliza o que é denominado de virtual Document Object Model (DOM), basicamente um objecto JavaScript utilizado para monitorizar e interagir em primeira linha, sendo alterado o DOM da página apenas quando é necessário, evitando esse processo demorado de constante localização e actualização dos elementos da página.

Podendo utilizar-se um modelo modular de desenvolvimento e reaproveitar funcionalidades exportadas nos ditos módulos, torna-se flexível na sua utilização.

Embora existam routers para adicionar à solução, estes não são mantidos pela Facebook, não sendo garantida a sua total compatibilidade ou continuidade de desenvolvimento sendo estes apenas disponibilizados pela comunidade. Da mesma forma para evitar o dito código esparguete, e facilitar a transição de informação entre módulos, existem alternativas ás

propriedades que este disponibiliza para esse mesmo efeito, embora estas soluções sejam também providenciadas pela comunidade como mencionado no ponto anterior.

*Figura 11 - Comparação de frameworks de desenvolvimento Mobile*

AngularJS actualmente na versão 6, uma solução bastante completa com várias funcionalidades que permitem estar pronta para a mais complexa ou mais simples das aplicações, de entre os componentes necessários apenas não inclui a gestão de estado, tem esta caso se queira implementar ser utilizado um módulo externo para esse efeito, sendo todo o resto incluindo o router mantido pela própria Google.

Embora não utilize DOM virtual, utiliza uma detecção de alterações ao nível dos modelos, podendo alterando então apenas as áreas que necessitam dessa dita alteração. Recorrendo também ao desenvolvimento por módulos, permite ter os mesmos benefícios que a solução anterior.

VueJs, no entanto, é um estilo de fusão de ambas as frameworks mencionadas anteriormente, sendo leve, no entanto, eficaz, embora não tenha o conjunto de funcionalidades tão vasto como o oferecido pela AngularJs, é uma framework com funcionalidade suficiente para a generalidade dos sites. Utilizando também a mesma técnica de utilização de um DOM virtual, monitoriza essas alterações e usufrui de um elevado desempenho como podemos verificar na Figura 12 retirada de [14]. Possui um router também desenvolvido pela própria equipe, bem como uma ferramenta para gestão de estado dedicada, VueX. Não existindo, no entanto, uma grande empresa a manter o seu desenvolvimento, não tendo sido impedimento para a crescente popularidade desta framework.

*Figura 12 - BenchMark de diversas frameworks JavaScript*

Duration in milliseconds ± standard deviation (Slowdown = Duration / Fastest)

| Name | vue-v2.5.3-keyed | angular-v5.0.0-no-zone-keyed | angular-v5.0.0-keyed | react-v16.1.0-keyed | react-v16.1.0-redux-v3.7.2-keyed | angular-light-v0.14.1-keyed | react-v16.1.0-mobX-v3.3.1-keyed | angular-v1.6.3-keyed | react-v16.1.0-easy-state-v3.0.1-keyed |
|---|---|---|---|---|---|---|---|---|---|
| create rows<br>Duration for creating 1000 rows after the page loaded. | 169.2 ± 3.6 (1.0) | 170.9 ± 6.4 (1.0) | 185.7 ± 7.8 (1.1) | 201.2 ± 12.1 (1.2) | 206.2 ± 9.0 (1.3) | 163.8 ± 9.4 (1.0) | 234.4 ± 7.3 (1.4) | 222.9 ± 8.1 (1.4) | 217.7 ± 8.7 (1.3) |
| replace all rows<br>Duration for updating all 1000 rows of the table (with 5 warmup iterations). | 161.8 ± 3.9 (1.0) | 176.6 ± 5.4 (1.1) | 179.3 ± 6.5 (1.1) | 169.0 ± 4.3 (1.0) | 175.1 ± 4.2 (1.1) | 175.1 ± 16.8 (1.1) | 193.4 ± 8.9 (1.2) | 232.3 ± 8.7 (1.4) | 194.0 ± 6.4 (1.2) |
| partial update<br>Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows. | 168.1 ± 7.4 (2.3) | 73.7 ± 4.1 (1.0) | 73.5 ± 4.9 (1.0) | 90.9 ± 3.3 (1.2) | 97.8 ± 5.6 (1.3) | 77.3 ± 3.4 (1.1) | 89.7 ± 2.5 (1.2) | 87.1 ± 5.3 (1.2) | 617.0 ± 21.5 (8.4) |
| select row<br>Duration to highlight a row in response to a click on the row. (with 5 warmup iterations). | 9.8 ± 2.5 (1.0) | 8.8 ± 3.5 (1.0) | 7.6 ± 4.0 (1.0) | 12.4 ± 4.1 (1.0) | 10.1 ± 3.5 (1.0) | 10.8 ± 3.9 (1.0) | 9.2 ± 5.3 (1.0) | 10.0 ± 4.7 (1.0) | 7.5 ± 3.7 (1.0) |
| swap rows<br>Time to swap 2 rows on a 1K table. (with 5 warmup iterations). | 19.0 ± 2.8 (1.0) | 117.9 ± 2.7 (6.2) | 118.5 ± 2.8 (6.2) | 121.8 ± 4.2 (6.4) | 121.7 ± 4.9 (6.4) | 117.0 ± 5.4 (6.2) | 126.7 ± 4.3 (6.7) | 125.9 ± 5.3 (6.6) | 128.1 ± 3.8 (6.8) |
| remove row<br>Duration to remove a row. (with 5 warmup iterations). | 52.5 ± 1.8 (1.2) | 43.5 ± 2.3 (1.0) | 46.1 ± 2.6 (1.1) | 51.5 ± 2.0 (1.2) | 49.5 ± 1.8 (1.1) | 133.5 ± 7.2 (3.1) | 52.5 ± 1.5 (1.2) | 48.6 ± 2.5 (1.1) | 56.5 ± 2.6 (1.3) |
| create many rows<br>Duration to create 10,000 rows | 1,521.4 ± 55.7 (1.0) | 1,629.6 ± 53.4 (1.1) | 1,682.0 ± 53.1 (1.1) | 2,033.7 ± 32.0 (1.3) | 2,048.5 ± 58.8 (1.3) | 1,657.2 ± 58.8 (1.1) | 2,337.9 ± 54.5 (1.5) | 2,112.0 ± 77.7 (1.4) | 2,218.9 ± 51.4 (1.5) |
| append rows to large table<br>Duration for adding 1000 rows on a table of 10,000 rows. | 338.4 ± 10.3 (1.3) | 275.4 ± 5.1 (1.1) | 257.6 ± 11.1 (1.0) | 271.8 ± 9.9 (1.1) | 300.7 ± 30.9 (1.2) | 293.8 ± 17.4 (1.1) | 367.2 ± 40.8 (1.4) | 371.6 ± 60.4 (1.4) | 427.5 ± 12.1 (1.7) |
| clear rows<br>Duration to clear the table filled with 10.000 rows. | 240.9 ± 11.4 (1.1) | 334.7 ± 25.1 (1.5) | 360.3 ± 16.4 (1.6) | 224.4 ± 6.0 (1.0) | 227.7 ± 8.6 (1.0) | 270.1 ± 5.2 (1.2) | 308.8 ± 25.8 (1.4) | 517.8 ± 62.0 (2.3) | 262.6 ± 9.8 (1.2) |
| slowdown geometric mean | 1.17 | 1.32 | 1.35 | 1.37 | 1.40 | 1.47 | 1.55 | 1.63 | 1.91 |

## 2.2 Testes

Ao investigar sobre os tipos de testes, pode-se constatar que existem inúmeros tipos de testes possíveis de realizar, sendo que estes são subdivididos em diversas categorias como por exemplo testes funcionais e não funcionais, existindo também diversas técnicas de testes.

### 2.2.1 Testes funcionais e não funcionais

Estes dois tipos de testes têm intuito bastante diferente embora relacionado, nos testes funcionais é efectuada a validação e verificação que todas as funcionalidades desenvolvidas correspondem aos requisitos do cliente, e conformidade do código sendo efectuados antes dos testes não funcionais. Os testes não funcionais por sua vez são então efectuados após a validação do código, sendo estes focados em questões de desempenho, validar por exemplo se um dashboard carrega a sua interface em x segundos, se um sistema é capaz de aguentar x utilizadores em simultâneo, etc. Podemos verificar uma versão comparativa resumida na Figura 13 retirada de [15].

**Vs funcionais. Teste não funcional**

Functional Testing VS Non-Functional Testing

| Parâmetros | Funcional | Teste não funcional |
|---|---|---|
| **Execução** | É realizado antes de testes não funcionais. | É realizado após o teste funcional. |
| **Área de foco** | É baseado nos requisitos do cliente. | Centra-se na expectativa do cliente. |
| **Requerimento** | É fácil definir requisitos funcionais. | É difícil definir os requisitos para testes não funcionais. |
| **Uso** | Ajuda a validar o comportamento do aplicativo. | Ajuda a validar o desempenho do aplicativo. |
| **Objetivo** | Realizada para validar ações de software. | Isso é feito para validar o desempenho do software. |
| **Requisitos** | O teste funcional é realizado usando a especificação funcional. | Esse tipo de teste é realizado por especificações de desempenho |
| **Teste manual** | O teste funcional é fácil de executar por testes manuais. | É muito difícil realizar testes não funcionais manualmente. |
| **Funcionalidade** | Descreve o que o produto faz. | Descreve como o produto funciona. |
| **Exemplo de caso de teste** | Verifique a funcionalidade de login. | O painel deve carregar em 2 segundos. |
| **Tipos de teste** | Exemplos de tipos de testes funcionais<br>• Teste unitário<br>• Teste de fumaça<br>• Aceitação do usuário<br>• Teste de integração<br>• Testes de regressão<br>• Localização<br>• Globalização<br>• Interoperabilidade | Exemplos de tipos de teste não funcionais<br>• Teste de performance<br>• Teste de Volume<br>• Escalabilidade<br>• Testando usabilidade<br>• Teste de Carga<br>• Teste de estresse<br>• Teste de conformidade<br>• Teste de portabilidade<br>• Teste de Recuperação de Desastres |

*Figura 13 - Testes funcionais Vs. não funcionais*

Com estes dois tipos de testes definidos e no âmbito do projecto, os testes relevantes e que foram focados são os testes não funcionais, pois quer-se determinar características relativamente a performance das diversas tecnologias, bem como não existe uma aplicação nem critérios definidos para utilização da mesma.

### 2.2.2 Principais tipos de testes não funcionais

Dentro dos testes não funcionais e segundo o glossário do International Software Testing Qualifications Board - ISTQB possível de consultar em https://glossary.istqb.org/ podemos encontrar os seguintes tipos de testes não funcionais como sendo os principais tipos de testes não funcionais possíveis de realizar

#### 2.2.2.1 Testes de performance

Permitem determinar o grau de eficácia com que um determinado sistema é capaz de dar resposta a certas situações com base em critérios como por exemplo tempos de resposta e rapidez de resposta.

#### 2.2.2.2 Testes de carga

Este teste tem como objectivo a medição do comportamento e desempenho de um componente ou sistema sobre carga incremental, como determinar o número máximo de utilizadores paralelos possível ou transacções concorrentes.

#### 2.2.2.3 Testes de stress

Um tipo de teste de performance que tem como objectivo a realização de testes acima do limiar para que a aplicação ou sistema foi desenhado ou com baixa disponibilidade de recursos.

#### 2.2.2.4 Testes de usabilidade

Tipo de teste utilizado para perante determinados critérios, avaliar a facilidade de utilização, compreensão e aprendizagem dos utilizadores bem como a atractividade da solução.

#### 2.2.2.5 Manutenção

Determinação do grau de dificuldade com que o sistema pode ser alterado para dar resposta a alterações, correcção de defeitos, modificação para abranger novos requisitos, modificação para facilitar futura manutenção e alteração de ambiente.

#### 2.2.2.6 Fiabilidade

Teste à capacidade do produto a executar as funções para as quais foi desenvolvido durante um determinado tempo.

#### 2.2.2.7 Portabilidade

Teste à capacidade de resposta do produto a alteração de ambiente.

### 2.2.3 Testes dinâmicos e testes estáticos

Achou-se também relevante mencionar a diferença entre os conceitos de testes dinâmicos e testes estáticos, constatando-se que os testes estáticos consistem numa revisão ou avaliação do código após o desenvolvimento do mesmo, mas antes da execução, um exemplo comum deste tipo de testes é o code review, em que um outro programador que não o que desenvolveu o código a analisar, faz uma vistoria de forma a garantir que as melhores práticas foram implementadas e o código se encontra conforme. Existem, no entanto, também algumas ferramentas de análise de código de forma a automatizar esse processo, identificando por exemplo leaks de memória, stack overflows ou mesmo concorrência, existindo aplicações especificas para cada linguagem, podemos constatar alguns exemplos das mesmas em [16].

Por outro lado, temos a análise de código dinâmica, que se poderá dizer que ocorre numa fase posterior, ou de uma forma um pouco oposta, neste tipo de testes o código é corrido e analisada essa execução bem como os resultados da mesma. Existindo também algumas ferramentas para cada linguagem que também podem ser consultadas em [16].

## 2.3 Escolha dos tipos de testes

Tendo em conta a inexistência de uma aplicação para testar e que o objectivo é uma análise prévia e comparativa de forma a obter um conjunto de testes enquadrados no objectivo da futura elaboração da aplicação na tecnologia escolhida com base no estudo aqui realizado, dentro destes testes nas várias subcategorias denotam-se os testes de performance que nos iram permitir medir a reactividade e tempos de espera na utilização das aplicações de teste desenvolvidas, bem como os testes de carga, que iram ser determinantes para examinar a performance da API.

# 3 Solução proposta

## 3.1 Identificação de pontos a testar

Tendo como base o problema apresentado, nomeadamente a escolha da tecnologia para elaboração de uma aplicação móvel no âmbito do projecto VITASENIOR, foram identificados os seguintes requisitos:

- Rapidez de execução das tarefas;
- Quantidade e gestão de memória alocada;
- Rapidez de comunicação com a API;
- Simplicidade e facilidade de manutenção do código.

Tendo em conta estes critérios, foram extrapoladas algumas funcionalidades e testes a correr no sistema, sendo definido que se deveria de testar a efectividade na utilização do CPU, efectuada a medição do consumo de memória, tempo de execução de operações habituais como o registo e login na aplicação e finalmente complexidade do código e sua consequente facilidade em manter.

Para que estes pontos pudessem ser analisados e quantificados de forma a efectuar uma comparação entre as diversas tecnologias seleccionadas, foram extrapoladas métricas e tabelas a preencher em cada um dos testes, efectuando a ligação caso de uso a resultado obtido para cada tecnologia a nível individual, de forma a se elaborar uma pequena conclusão relativamente ao seu desempenho nas referidas tarefas.

No final todos os dados serão concentrados numa única tabela de forma a extrapolar resultados finais, sendo depois seleccionadas duas tecnologias para a realização de duas mini aplicações por forma a realizar-se uma comparação de usabilidade entre ambas.

Sendo que não existia nenhum tipo de aplicação móvel desenvolvida, foram então planeadas tarefas a desenvolver para os casos acima mencionados, sendo seguidamente mencionado o tópico e respectivas funcionalidades a desenvolver.

Rapidez de execução das tarefas:

- Execução repetida de operação de registo e login;

- Tarefa de execução intensiva do CPU em single thread;
- Tarefa de execução intensiva do CPU em multi thread.

Quantidade e gestão de memória alocada:

- Navegação repetida de forma a criar novas referencias de memória.

Rapidez de comunicação com a API:

- Realização de chamadas repetidas e encadeadas a diversos métodos da API

Simplicidade e facilidade de manutenção do código:

- Análise comparativa final entre quantidade e complexidade do código realizado.

## 3.2 Selecção de tecnologias

Para que se pudesse realizar a comparação foi decidido efectuar-se o desenvolvimento de três aplicações de teste com as funcionalidades acima descritas, partindo do principio que o código desenvolvido em cada uma dessas mesmas aplicações deveria tentar manter-se o mais aproximado possível, sendo que existiram sempre algumas alterações devido a especificidades tanto das plataformas como das linguagens de desenvolvimento utilizadas.

A selecção recaiu então sobre, elaboração de uma PWA, sendo este um conceito relativamente recente e em crescente adopção, com particularidade de se poder instalar no dispositivo e ter comportamento semelhante a uma aplicação nativa, bem como a capacidade de ser utilizada sem conexão de rede e mesmo ser possível o acesso a várias funcionalidades nativas do dispositivo.

Elaboração de uma aplicação nativa como base comparativa para as restantes tecnologias, pois se estamos a efectuar um estudo comparativo deve estar presente a base, neste caso umas das bases, nomeadamente uma aplicação para sistema operativo Android devido ao facto da não disponibilidade de um dispositivo MAC OS para realização do desenvolvimento, pois este é um requisito para ter acesso à ferramenta XCode necessária para o desenvolvimento de aplicações para a plataforma proprietária da Apple, iOS.

E finalmente para representação de uma das plataformas de desenvolvimento hibrido, foi seleccionada Ionic, pois esta apresenta um bom desenvolvimento e acompanhamento das

tecnologias, possuindo também já, embora em fase beta um PWA Toolkit, que permite o desenvolvimento de Progressive Web Apps, embora se irá focar apenas neste momento na realização de um projecto para exportação para Android uma vez que queremos comparar com a plataforma mencionada.

## 3.3 Hardware e largura de banda

Sendo tanto para a realização dos testes como para a execução do servidor para a PWA importante o hardware utilizado, todos os testes foram realizados recorrendo ao hardware mencionado de seguida.

Hardware para testes à API e hospedagem do servidor da PWA:

- CPU: i7-8750H 8ª Geração Intel® Core™
- RAM: 2 x 16GB G.SKILL RIPJAWS DDR4 2400Mhz
- GPU: Nvidia GeForce GTX 1060 6GB 10ª Geração
- DISCO: 2 x 256GB Samsung 970 EVO NVMe M.2 RAID 0
- Placa de rede wireless: KILLER AC 1550 WIRELESS LAN + BT 5.0
- Placa de rede: Realtek PCIe GBE Family Controller (10/100/1000M Gigabit Ethernet)

Hardware de testes das aplicações móveis:

- HUAWEI Mate 10 Lite

Largura de banda:

- Os testes foram efectuados com uma largura de banda de 100Mbps simétricos dedicados.

*Figura 14 – Teste realizado em : http://speedmeter.fccn.pt/*

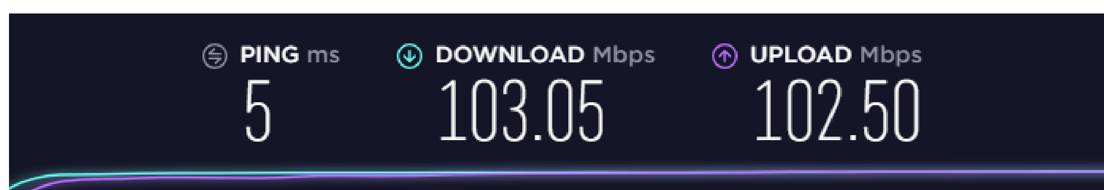



*Figura 15 – Teste realizado em: http://www.speedtest.net/pt*

## 3.4 Testes à API

De forma a que se possa obter testes viáveis e correctos, podendo afirmar com algum grau de certeza que a existência de alguma diferença de performance se deve apenas às implementações/soluções seleccionadas, foram levados a cabo alguns testes de performance à API que irá ser utilizada, sendo esta a API que alimenta o projecto VITASENIOR.

Após a elaboração de uma investigação a ferramentas possíveis de ser utilizadas para realizar este tipo de testes a API mencionada, foi seleccionada a aplicação JMeter, que é tida como uma aplicação de referencia entre as soluções Open Source para teste a API's, sendo dotada de uma boa interface com bastantes funcionalidades, fácil utilização e boa eficácia [17, 18, 19, 20]

Para determinar o limiar de pedidos possíveis de realizar foram definidos alguns critérios que deveriam ser respondidos para se chegar a uma conclusão assertiva, sendo estes:

1. Quantos pedidos o hardware de testes é capaz de fazer?
2. Quantos pedidos por segundo é a instancia capaz de dar resposta?
3. Quantos pedidos por segundo é a API capaz de dar resposta?
4. A capacidade de resposta da API é igual para todas as rotas?

Sendo as abordagens tomadas e respectivos resultados e sua breve análise demonstrados nos subtópicos seguintes.

### 3.4.1 Quantos pedidos o hardware de teste é capaz de fazer?

Para dar resposta a esta questão foi efectuado um teste de carga sem conectividade à rede, tendo como objectivo a falha imediata dos mesmos, quantificando-se assim apenas a capacidade de o hardware iniciar os pedidos.

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughp... | Received... | Sent KB/... | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Register | 71628 | 0 | 0 | 73 | 0,37 | 100,00% | 7552,5/s... | 14389,62 | 0,00 | 1951,0 |
| TOTAL | 71628 | 0 | 0 | 73 | 0,37 | 100,00% | 7552,5/s... | 14389,62 | 0,00 | 1951,0 |

*Figura 16 – Máximo de pedidos registado sem conexão de rede activa*

Inicialmente foram registados máximos a rondar os 7500 pedidos por segundo, sendo que este valor decresceu um pouco com o tempo estabilizando em cerca de 3000 pedidos por segundo.

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughp... | Received... | Sent KB/... | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Register | 329758 | 0 | 0 | 2489 | 6,10 | 100,00% | 3082,7/s... | 5873,49 | 0,00 | 1951,0 |
| TOTAL | 329758 | 0 | 0 | 2489 | 6,10 | 100,00% | 3082,7/s... | 5873,49 | 0,00 | 1951,0 |

*Figura 17 – Média de pedidos por segundo registados*

Sendo que no caso dos 7500 pedidos estavam a ser utilizados a quase totalidade de CPU do computador e no período da média estável registou-se um consumo de recursos mais baixo.

Com a análise dos resultados obtidos podemos concluir que o hardware utilizado suporta a execução de pelo menos 3000 pedidos por segundo.

### 3.4.2 Quantos pedidos por segundo a instancia é capaz de dar resposta?

Para a análise deste ponto, foi determinado a realização de um teste de carga para uma rota não reconhecida pela api, tentando evitar a interferência na resposta por parte do código da própria API, ficando apenas os tempos de resposta relacionados com a instancia onde a API se encontra alojada, framework e router utilizados bem como as condições de rede.

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| FakeRegister | 2000 | 2230 | 381 | 4273 | 1132,28 | 100,00% | 384,2/sec | 199,86 | 125,50 | 532,6 |
| TOTAL | 2000 | 2230 | 381 | 4273 | 1132,28 | 100,00% | 384,2/sec | 199,86 | 125,50 | 532,6 |

*Figura 18 – Máximo de pedidos à rota desconhecida registados*

Após a realização de vários testes podemos verificar que os valores médios dos pedidos se encontravam aproximadamente entre os 350 e os 384 pedidos por segundo, sendo este ultimo o máximo registado.

### 3.4.3 Quantos pedidos por segundo a API é capaz de dar resposta?

De forma a dar resposta a este ponto foram registados testes de carga em circunstancias normais, sendo a rota seleccionado o de registo na API, para a realização deste teste foi utilizado um CSV com os parâmetros necessário de forma a que todos os pedidos efectuados tivessem a mesma resposta, encontrando-se em total igualdade de circunstancias.

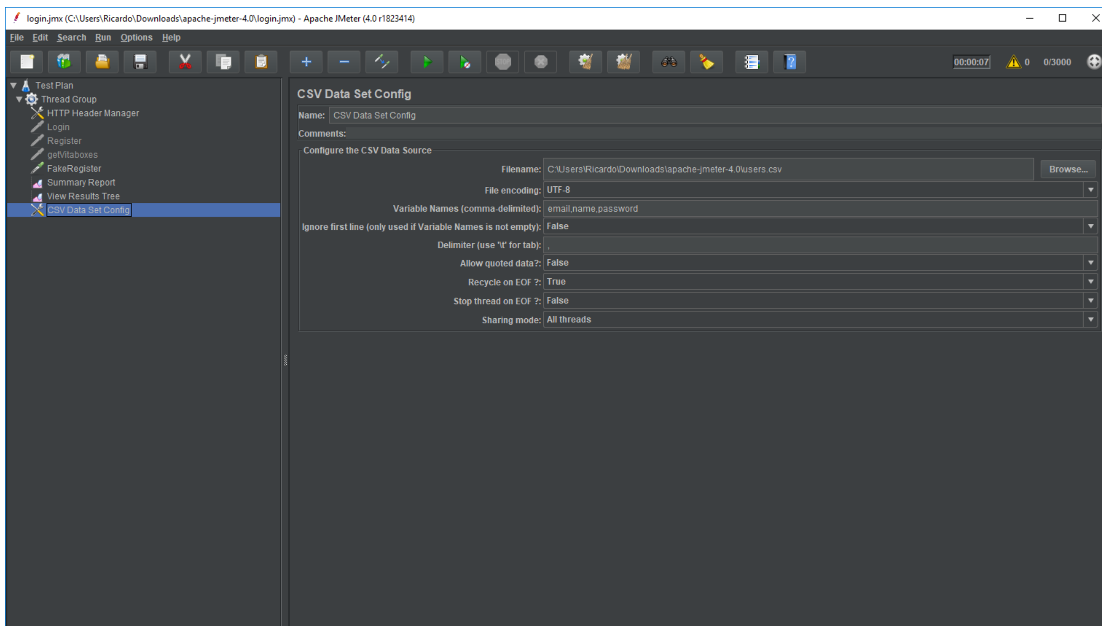



*Figura 19 – Utilização de ficheiro CSV com campos necessários ao pedido*

Sendo o preenchimento dos parâmetros no corpo do pedido efectuado através de um simples mapeamento do nome verificado no exemplo da Figura 19, utilizado da forma demonstrada na Figura 20.

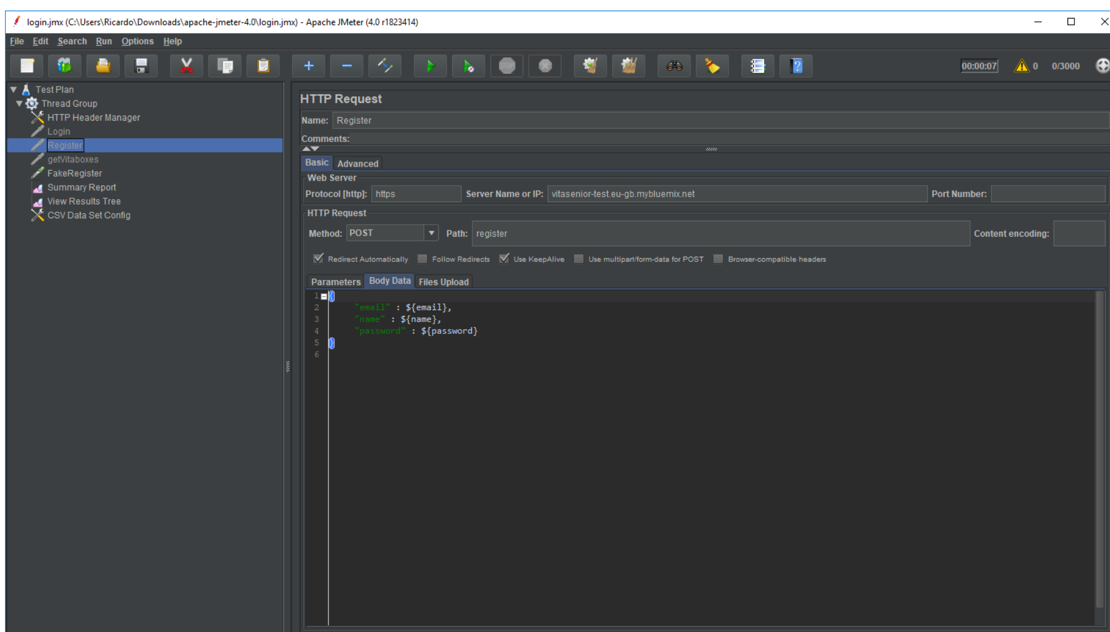



*Figura 20 – Utilização de parâmetros mapeados do ficheiro CSV*

Neste ponto os resultados obtidos rondam os 50 pedidos por segundo como demonstrado na Figura 21.

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Register | 2000 | 17840 | 373 | 29904 | 9176,90 | 0,00% | 50,4/sec | 55,26 | 16,25 | 1123,4 |
| TOTAL | 2000 | 17840 | 373 | 29904 | 9176,90 | 0,00% | 50,4/sec | 55,26 | 16,25 | 1123,4 |

*Figura 21 – Pedidos por segundo para rota registar*

### 3.4.4 A capacidade de resposta da API é igual para todos as rotas?

Relativamente a este ponto a solução realizada foi efectuar o mesmo tipo de deste que foi efectuado anteriormente, mas para outras rotas da API de forma a poder verificar a existência ou não de diferentes performances nas diferentes rotas.

Para a realização deste teste foram seleccionadas mais 3 rotas diferentes, sendo seleccionadas rotas que iram ser vulgarmente utilizadas na aplicação, sendo estas o Login, List e Get Users, podendo os resultados obtidos ser verificados na Figura 22,Figura 23 e Figura 24.

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Login | 2000 | 14994 | 1117 | 23485 | 4760,50 | 0,00% | 65,7/sec | 74,51 | 52,24 | 1160,9 |
| TOTAL | 2000 | 14994 | 1117 | 23485 | 4760,50 | 0,00% | 65,7/sec | 74,51 | 52,24 | 1160,9 |

*Figura 22 – Pedidos por segundo médios registado nas operações de login*

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| getVitaboxes | 2000 | 5331 | 443 | 9252 | 2452,40 | 0,00% | 129,7/sec | 112,55 | 95,21 | 888,9 |
| TOTAL | 2000 | 5331 | 443 | 9252 | 2452,40 | 0,00% | 129,7/sec | 112,55 | 95,21 | 888,9 |

*Figura 23 – Pedidos por segundo médios registados nas operações de Listagem de VITABOX's*

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received ... | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| getUsers | 3000 | 18940 | 2913 | 32470 | 6582,54 | 0,00% | 89,4/sec | 57,16 | 68,50 | 655,0 |
| TOTAL | 3000 | 18940 | 2913 | 32470 | 6582,54 | 0,00% | 89,4/sec | 57,16 | 68,50 | 655,0 |

*Figura 24 – Pedidos por segundo médios registados nas operações de listagem de utilizadores de uma box.*

Entre as várias conjugações de utilizadores e números de pedido por segundo efectuados na ferramenta podemos verificar que a API suporta inicialmente um número de pedidos superior, após esse testes inicial caso este seja novamente executado, ocorre uma diminuição da capacidade de resposta, podendo-se dever isto a algum tipo de restrição imposta no servidor para protecção contra ataques de negação de serviço (do inglês Denial Of Service – DOS)/ negação de serviço distribuída (do inglês Distributed Denial Of Service – DDOS), sendo de importância referir que não foi possível efectuar os testes com conhecimento e ambiente diferente do ambiente normal de produção do prestador do serviço, sendo por tal facto a provável restrição verificada.

Analisando então os resultados obtidos dos vários testes a várias rotas, constatou-se que existem diferentes tempos de resposta e respectivo aumento ou diminuição dos pedidos por segundo suportados. Com estes resultados foram então definidos máximos de teste para as aplicações móveis realizadas quando o teste envolver pedidos de rede à API, sendo removida uma margem de "segurança" de forma a não existir interferência da API num momento em que possa não obter o mesmo desempenho máximo.
Os limites máximos de pedidos definidos foram então os seguintes:

- 30 pedidos por segundo para operações de registo
- 50 pedidos por segundo para operações de login
- 100 pedidos por segundo para operações de listagem de VITABOX's
- 70 pedidos por segundo para operações de listagem de utilizadores das vitaboxes.

Com estes valores definidos foram então iniciados os testes às aplicações cliente realizadas.

## 3.5 Aplicações de Teste

Após todo o processo de selecção das tecnologias a utilizar e tipos de testes a realizar, foram então desenvolvidas e testadas as aplicações.
Mediante o acima descrito foram então desenvolvidas as três aplicações em Android, uma PWA com utilização da framework VueJs e uma outra em Ionic com recurso à framework Angular.

De forma a aferir os resultados obtidos, os mesmos foram realizados uma primeira vez para controlo, de forma a que processos internos das framework, colocações de objectos em

memória, ou outros possíveis factores que interferissem nos tempos de execução, não fossem tidos em conta. De forma a obter dados significativos e extrapolar um valor médio bem como um desvio padrão para que se pudesse aferir não só a rapidez como também a fiabilidade da operação, todos os testes realizados foram executados cinco vezes alem da execução de controlo.

Relativamente aos testes de rede, para alem de se anular o valor inicialmente superior verificado na realização dos testes à API com a execução do teste de controlo, foram também tidos em conta os resultados dos registos máximos das rotas, sendo então assim realizados estes testes de performance com repetições dentro dos valores mencionados a cima. No caso do teste de pedidos de rede interligados, foi utilizado o valor mais baixo das rotas, ignorando o valor correspondente ao login efectuado para atribuição de um token antes da execução dos restantes pedidos, uma vez que este é apenas realizado uma vez com esse mesmo intuito.

Relativamente à PWA e tendo em conta que o browser irá interferir nos resultados, pois cada browser terá a sua performance e codificação para alem de a base ser em alguns casos diferentes como o caso do chrome e do firefox, foram então efectuados os testes nos três principais browsers Chrome, Firefox e Opera.

### 3.5.1 Medição de tempos de acesso para Login / Registo

Como exemplo de uma operação de rotina que seria utilizada diríamos de forma quase diária, foram medidos os tempos de resposta da operação de login e registo. A cronometragem dos tempos inicia-se na acção de clicar no botão para efectuar login ou registo até ao momento da resposta.

Tendo em conta os limites mencionados foram então elaborados os testes cujos resultados são demonstrados nos pontos seguintes correspondentes a cada uma das tecnologias testadas.

#### 3.5.1.1 Android

Para a execução dos pedidos de rede foi utilizada a framework retrofit, sendo todo o processo entre o inicio do pedido até à resposta da "responsabilidade" da mesma, o resultado foi

medido a quando do inicio do processo até à recepção da resposta no listener apropriado, sendo contabilizado o número de pedidos efectuados, e registo do tempo total a quando do último pedido recebido.

Relativamente à rota de login, foram efectuadas as medições sendo então registados os respectivos tempos totais de acesso verificados na Tabela 1 e na Figura 25.

*Tabela 1 – Tempos registados para rota login (Android)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 1737 | 2428 | 1854 | 2282 | 2289 | 2471 | 2264,8 | 218,5364043 |

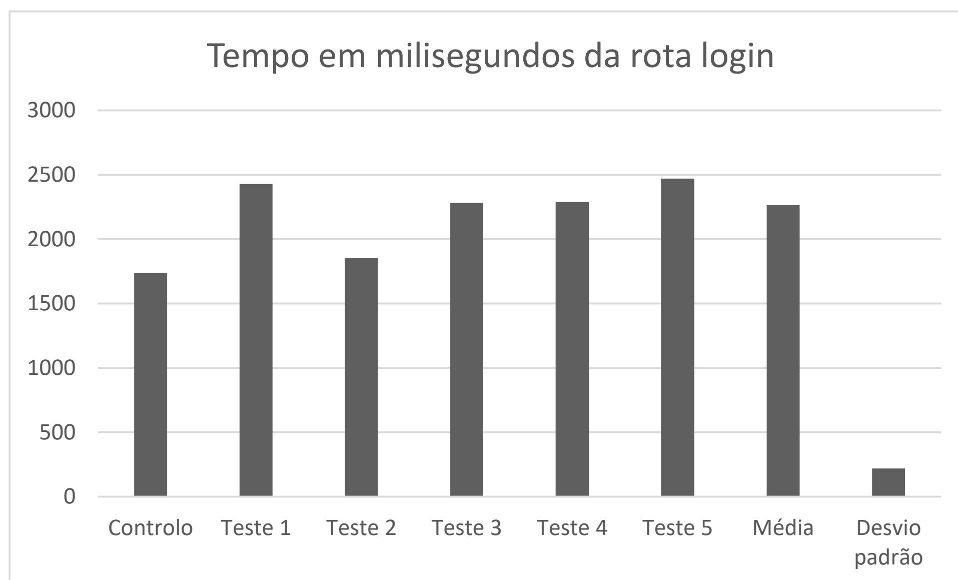



*Figura 25 – Representação gráfica dos resultados obtidos para rota login (Android)*

Analisando brevemente os resultados podemos observar que os tempos registados são bastante aceitáveis, tendo-se obtido um tempo médio por pedido de aproximadamente 45,3 milissegundos – ms, apresentando um desvio padrão de cerca de 4,37 ms, indicando que os tempos de resposta são consistentes, apresentando apenas desvios de aproximadamente 10% do valor médio.

Relativamente à rota de registar foi efectuado o mesmo processo, sendo então registados os resultados que se podem verificar na Tabela 2 e na Figura 26.

*Tabela 2 - Tempos registados para rota registar (Android)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 1135 | 1329 | 740 | 1136 | 819 | 1307 | 1066,2 | 244,7181236 |

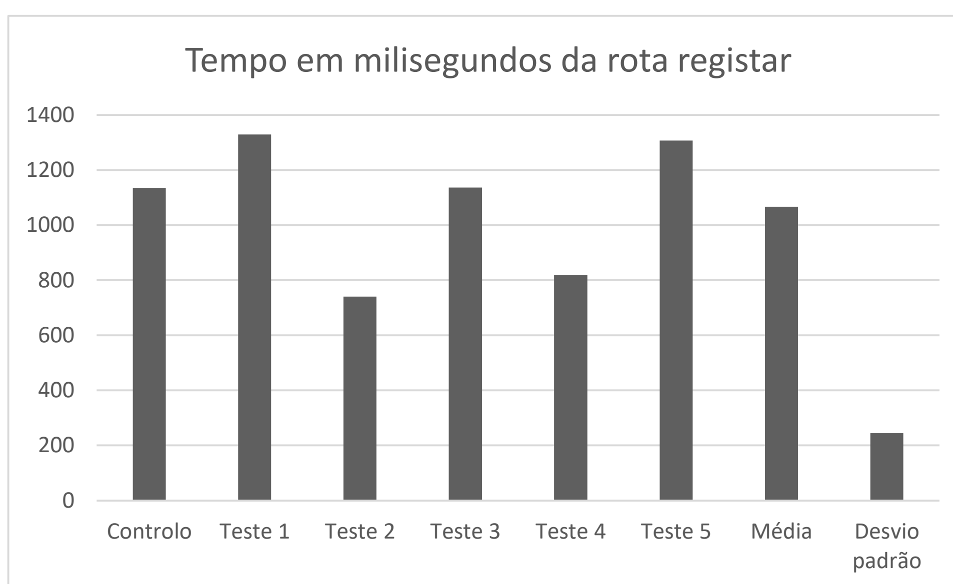



*Figura 26 - Representação gráfica dos resultados obtidos para rota registar (Android)*

Apresentando tempos de resposta médios de aproximadamente 36 ms, apresenta um resultado aceitável, podendo dar ao utilizador uma noção de execução instantânea. Neste caso, no entanto, o desvio padrão é bastante mais elevado, sendo de aproximadamente 8ms, representando cerca de 23% do valor médio. Perante estes resultados e tendo em conta a discrepância nomeadamente dos valores dos testes 2 e 4, o teste foi repetido por diversas vezes, sendo que continuou a dar valores significativamente diferentes a cada execução, não se podendo considerar os tempos de resposta obtidos muito consistentes.

#### 3.5.1.2 PWA

Relativamente à PWA os pedidos foram efectuados com recurso ao módulo axios, sendo o pedido igualmente registado no inicio do processo e contabilizado o tempo total na recepção da promessa do ultimo pedido recebido.

Nos testes realizados para a rota de login foram registados os seguintes tempos demonstrados na Tabela 3.

*Tabela 3 – Tempos registados para rota login (PWA)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Chrome | 2386 | 1819 | 1715 | 1750 | 1316 | 1944 | 1708,8 | 211,42 |
| Firefox | 2596 | 2642 | 1539 | 2080 | 1649 | 1475 | 1877 | 436,87 |
| Opera | 2462 | 2176 | 1653 | 1731 | 1584 | 1212 | 1671,2 | 308,93 |

Na sua representação gráfica presente na Figura 27 podemos facilmente verificar que se obtiveram melhores resultados nos browsers Chrome e Opera, tendo como verificado na Tabela 3 Opera obtido resultados ligeiramente melhores embora os resultados de android sejam ligeiramente mais consistentes.

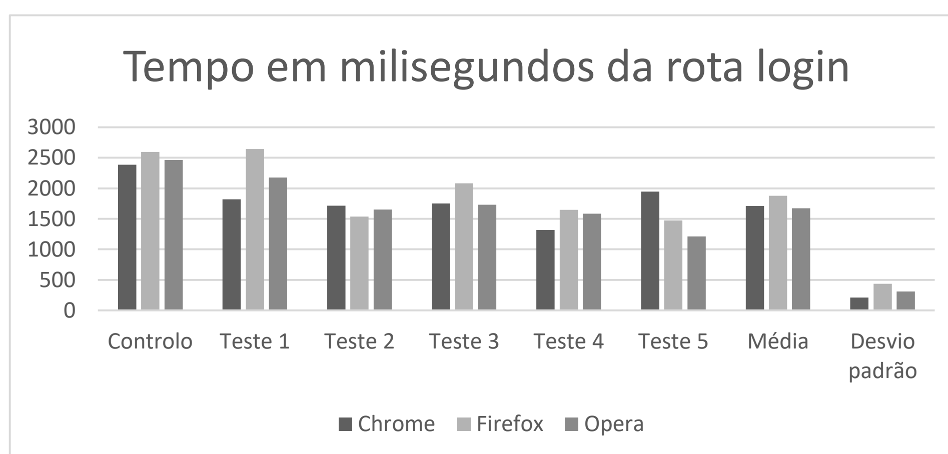



*Figura 27 – Representação gráfica dos resultados obtidos para rota login (PWA)*

Para verificar existências de diferenças de performance entre versões online e versões instadas foram novamente executados os mesmos testes para esse caso, obtendo os registos presentes na Tabela 4 e na Figura 28, verificando-se assim as mencionadas diferenças

*Tabela 4 - Tempos registados para rota login (PWA Instalada)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Instalada Chrome | 2208 | 1522 | 1487 | 1192 | 1736 | 2096 | 1606,6 | 299,87 |
| Instalada Firefox | 5213 | 2923 | 1923 | 1950 | 1829 | 1749 | 2074,8 | 430,03 |
| Instalada Opera | 2625 | 2126 | 2089 | 1753 | 1726 | 1552 | 1849,2 | 222,21 |

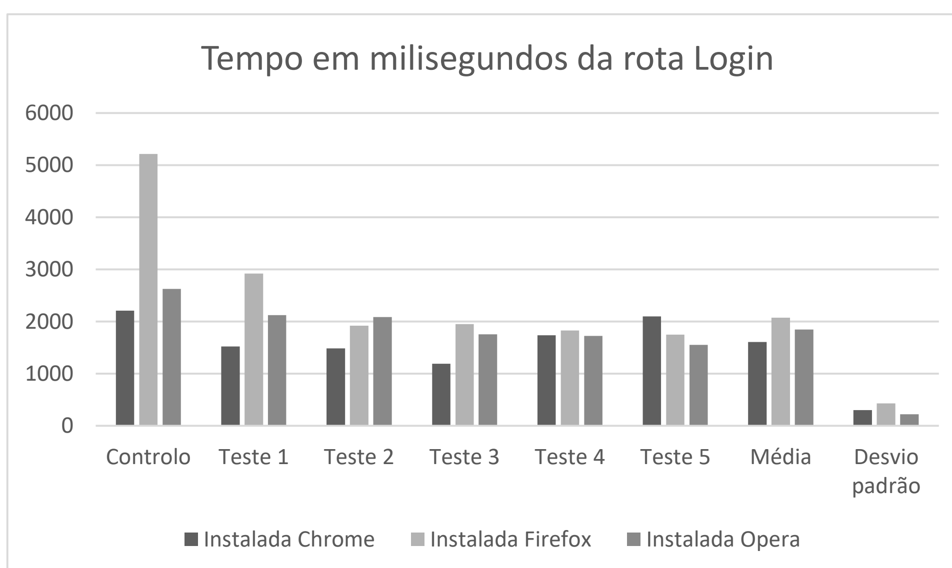



*Figura 28 - Representação gráfica dos resultados obtidos para rota login (PWA Instalada)*

Relativamente aos registos referentes à rota registar verificados na Tabela 5 - Tempos registados para rota registar (PWA)Tabela 5 e na Figura 29, podemos verificar que os resultados são mais homogéneos entre os diversos browsers apesar de se verificar um ligeiro desvio nos resultados obtidos.

*Tabela 5 - Tempos registados para rota registar (PWA)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Chrome | 1208 | 982 | 1311 | 1046 | 847 | 812 | 999,6 | 177,72 |
| Firefox | 881 | 952 | 912 | 859 | 1068 | 1273 | 1012,8 | 147,15 |
| Opera | 787 | 870 | 1213 | 923 | 1101 | 974 | 1016,2 | 124,71 |

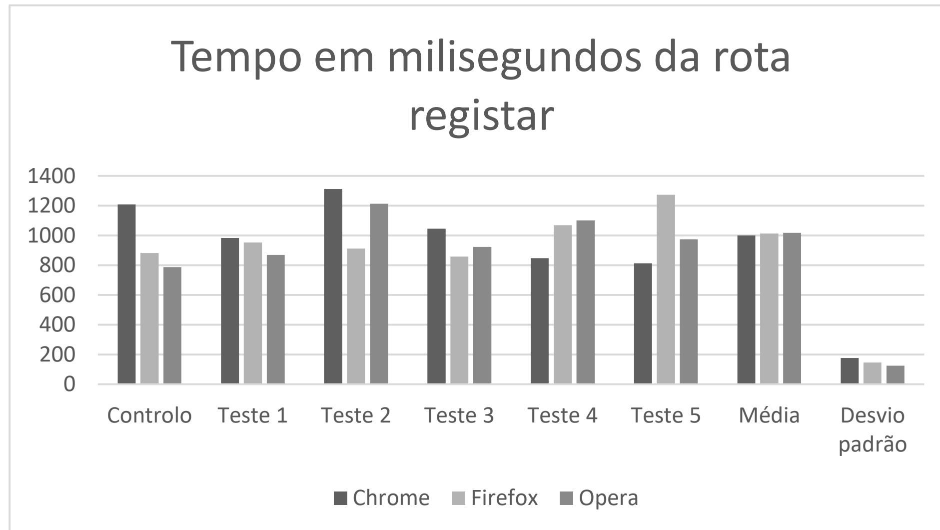



*Figura 29 - Representação gráfica dos resultados obtidos para rota registo (PWA)*

Da mesma forma que no anterior foi também foram efectuados os testes também para as versões instaladas nos dispositivos tendo os resultados sido os presentes na Tabela 6 e na Figura 30, podendo verificar-se novamente alguma diferença em relação aos tempos entre as versões instaladas e as versões a correr apenas no site.

*Tabela 6 - Tempos registados para rota registar (PWA Instalada)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Instalada Chrome | 1264 | 1136 | 1107 | 911 | 891 | 1077 | 1024,4 | 102,66 |
| Instalada Firefox | 1420 | 1146 | 1088 | 1159 | 998 | 1379 | 1154 | 126,02 |
| Instalada Opera | 1019 | 1054 | 951 | 1551 | 1109 | 945 | 1122 | 223,34 |

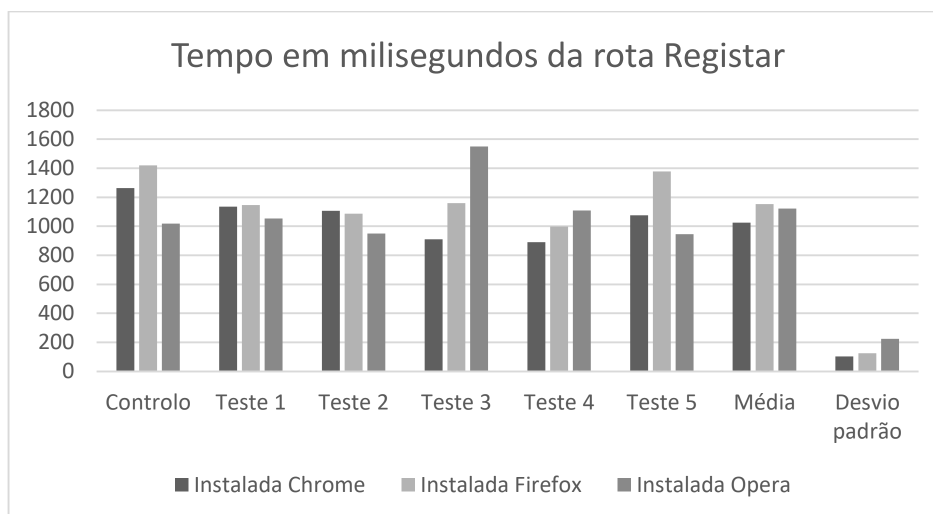



*Figura 30 - Representação gráfica dos resultados obtidos para rota registo (PWA instalada)*

#### 3.5.1.3 IONIC

Os pedidos foram efectuados através de um serviço que foi criado em que existe um método para realizar pedidos post e outro para get recorrendo ao módulo de Http.

Relativamente ao login procedeu-se à medição dos tempos tendo-se registados os valores contantes da Tabela 7 e Figura 31.

*Tabela 7 - Tempos registados para rota login (IONIC)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 3110 | 1999 | 2100 | 1015 | 1646 | 1157 | 1583,4 | 435,5257053 |

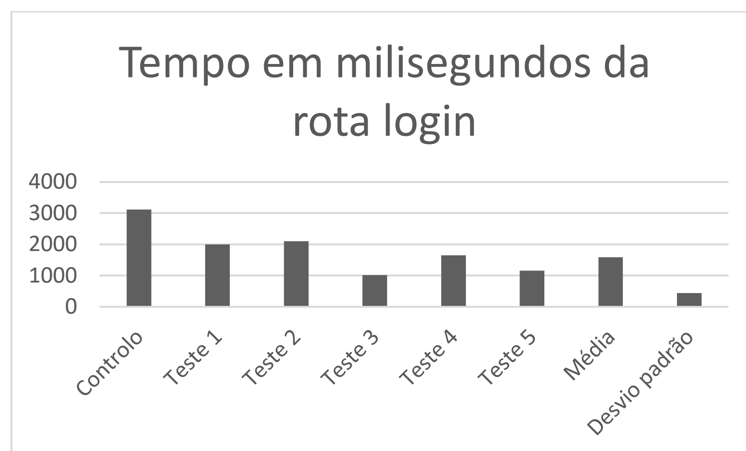



*Figura 31 - Representação gráfica dos resultados obtidos para rota login (IONIC)*

Com resultados médios por pedido de 38.57 ms e um desvio de aproximadamente 14.52 ms que corresponde a cerca de 37.64 por centro podemos verificar que apesar de os resultados se encontrarem em limites aceitáveis, mas com uma grande irregularidade nos diversos pedidos.

Relativamente ao registar foram efectuadas as medições, verificando-se os resultados presentes na Tabela 8 e na Figura 32.

*Tabela 8 - Tempos registados para rota registar (IONIC)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 1442 | 876 | 980 | 1487 | 901 | 973 | 1043,4 | 225,4139304 |

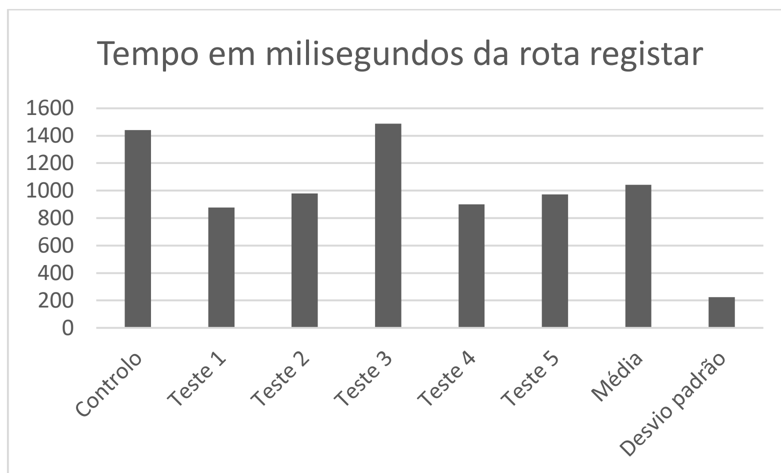



*Figura 32 - Representação gráfica dos resultados obtidos para rota registar (IONIC)*

Os resultados apresentaram um tempo médio de resposta de 34.78 ms e um desvio de 7.51 ms representando uma percentagem de cerca de 21.6 por cento, os resultados estão como os anteriores em valores aceitáveis, embora se verifique um desvio significativo, encontra-se novamente o teste três com um valor superior aos restantes, sendo de qualquer modo de esperar alguma irregularidade nos tempos dos pedidos.

### 3.5.2 Carregamento do armazenamento local

Para este teste foi utilizada uma imagem com cerca de 224Kb extensão JPEG e resolução de 2000 pixels por 1153 pixels.

De forma a que se obtivessem tempos minimamente significativos, mas ao mesmo tempo se ficasse dentro do limite de utilização de memória da aplicação, foi definido um limite de vinte mil repetições.

#### 3.5.2.1 Android

*Tabela 9- Tempos de carregamento de 100 imagens de armazenamento interno (Android)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 293 | 76 | 69 | 59 | 66 | 66 | 67,2 | 5,491812087 |

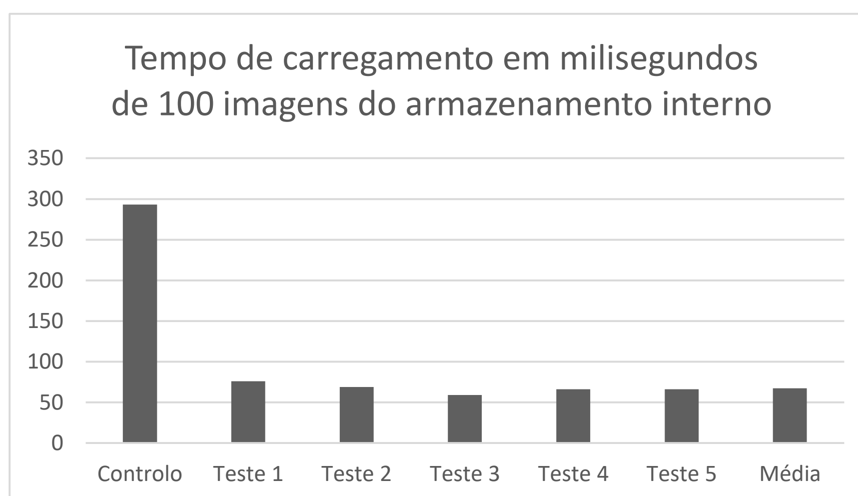



*Figura 33 – Representação gráfica dos resultados obtidos no carregamento de 100 imagens de armazenamento interno (Android)*

Relativamente ao carregamento de imagens do armazenamento interno podemos verificar tempo bastantes bons, sendo cada imagem em média carregada em menos de 1ms, e esta performance demonstra ser constante, pois apresenta um desvio padrão quase nulo em cada carregamento.

#### 3.5.2.2 PWA

Para o caso do carregamento de imagens na PWA o processo de carregamento é efectuado através do método normal de carregamento de imagens da framework, sendo o caminho para a imagem um caminho local.

Tendo então sido efectuada a medição dos tempos totais de carregamento e respectivo registo como podemos verificar na Tabela 10 e na Figura 34.

*Tabela 10 – Tempos de carregamento de 100 imagens de armazenamento interno (PWA)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Chrome | 183 | 101 | 89 | 108 | 111 | 102 | 102,2 | 7,57 |
| Firefox | 457 | 359 | 322 | 315 | 292 | 258 | 309,2 | 33,45 |
| Opera | 182 | 105 | 116 | 107 | 105 | 95 | 105,6 | 6,68 |

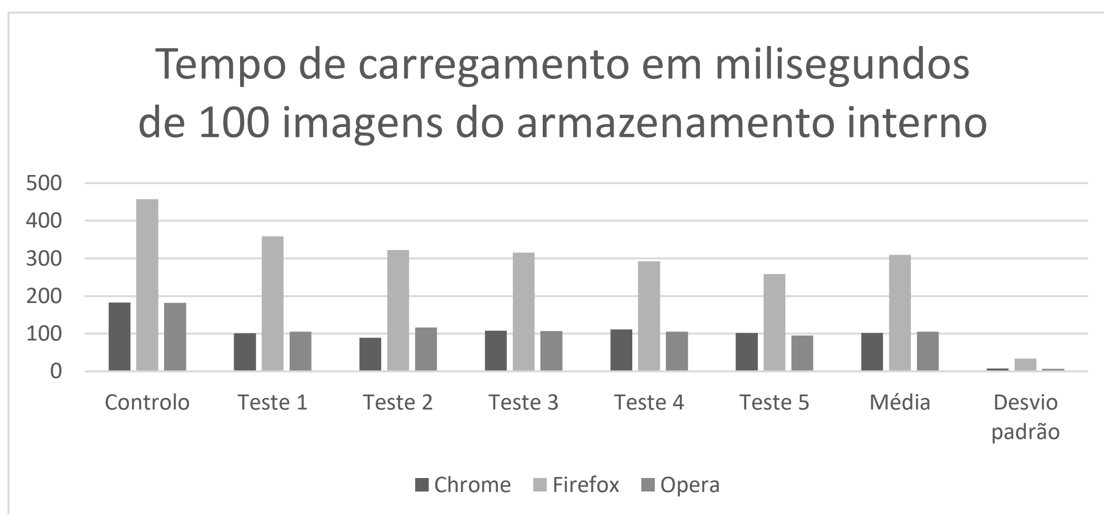



*Figura 34 – Representação gráfica dos resultados obtidos no carregamento de 100 imagens do armazenamento interno (PWA)*

Numa breve análise destes resultados podemos verificar uma grande discrepância entre os resultados obtidos no Firefox e os restantes, sendo que os resultados do Opera e Chrome são relativamente iguais, sendo também bastante consistentes com um desvio bastante baixo.

Para determinar se existe diferenças nos tempos entre as versões instaladas pelos respectivos browsers foram então repetidas as mesmas acções desta vez com recurso às versões mencionadas podendo os resultados ser verificados na Tabela 11 e na Figura 35.

*Tabela 11 - Tempos de carregamento de 100 imagens de armazenamento interno (PWA instalada)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Instalada Chrome | 1059 | 90 | 93 | 119 | 105 | 95 | 100,4 | 10,58 |
| Instalada Firefox | 457 | 362 | 300 | 364 | 364 | 346 | 347,2 | 24,55 |
| Instalada Opera | 329 | 93 | 95 | 98 | 104 | 95 | 97 | 3,85 |

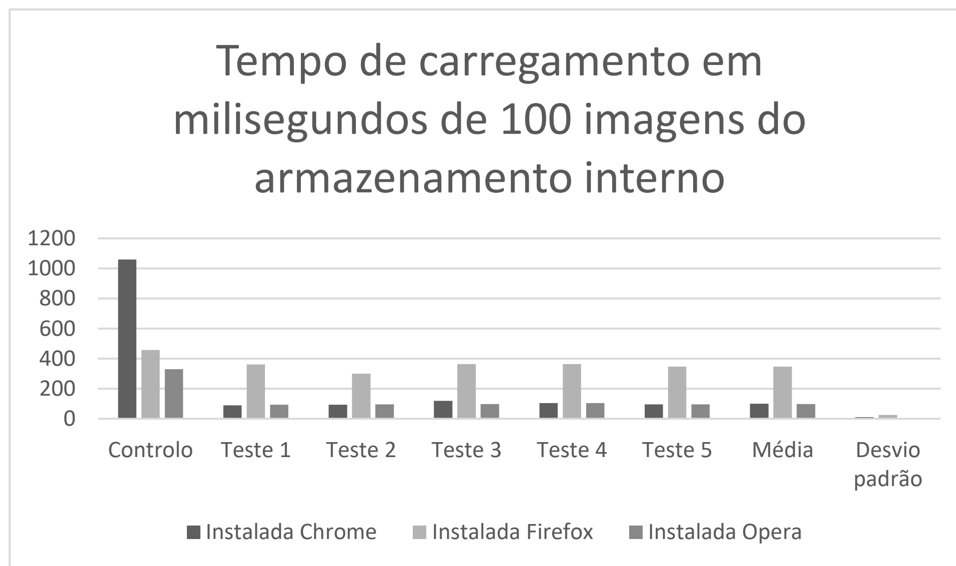



*Figura 35 - Representação gráfica dos resultados obtidos no carregamento de 100 imagens do armazenamento interno (PWA instalada)*

Podendo neste caso verificar que além do tempo de controlo no Chrome, os resultados são bastante semelhantes quer a nível de desvio como tempos de execução.

#### 3.5.2.3 IONIC

No caso desta framework foi utilizado o modo de carregamento normal da framework Angular para carregamento de imagem sendo o link da imagem correspondente a imagem da pasta de conteúdos estáticos.

*Tabela 12 - Tempos de carregamento de 100 imagens de armazenamento interno IONIC*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 649 | 704 | 695 | 685 | 650 | 670 | 680,8 | 19,09345438 |

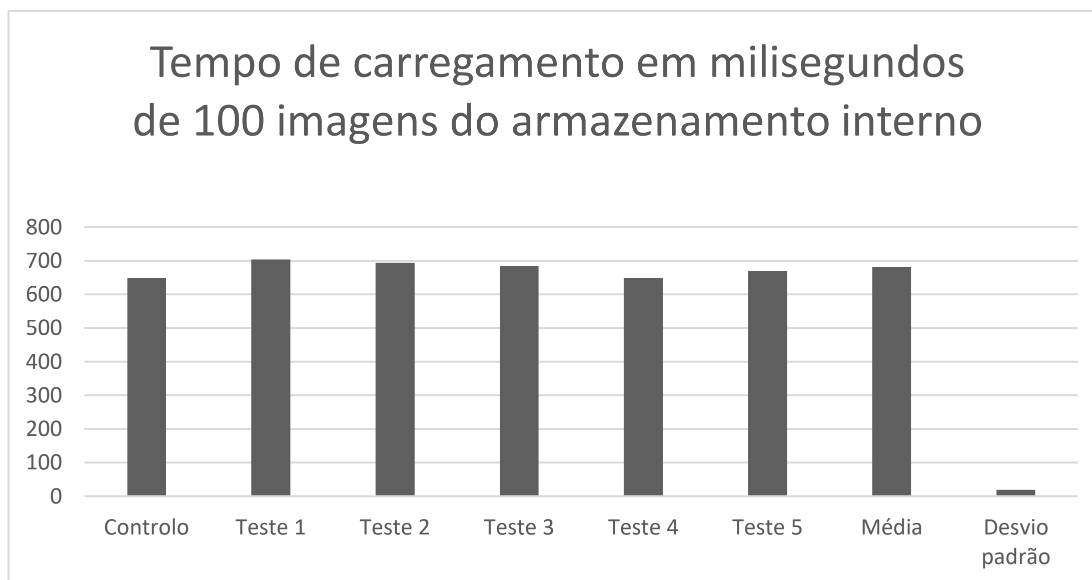



*Figura 36 - Representação gráfica dos resultados obtidos no carregamento de 100 imagens de armazenamento interno (IONIC)*

Neste exemplo podemos verificar o tempo de carregamento médio por imagem de aproximadamente 6.8 ms e um desvio de aproximadamente 0.19 ms correspondente a 2.80 por cento, verificando-se tempos aceitáveis bem como um desvio bastante baixo, correspondente a resultados homogéneos.

### 3.5.3 Carregamento por endereço da internet

#### 3.5.3.1 Android

Para a realização desta tarefa recorreu-se a uma AsyncTask para efectuar o download da imagem, sendo de seguida carregada para o componente que ira então ser carregado no ecrã.

*Tabela 13 - Tempos de carregamento de 25 imagens de endereço da internet (Android)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 4022 | 2839 | 3191 | 2833 | 2983 | 2856 | 2940,4 | 136,79 |

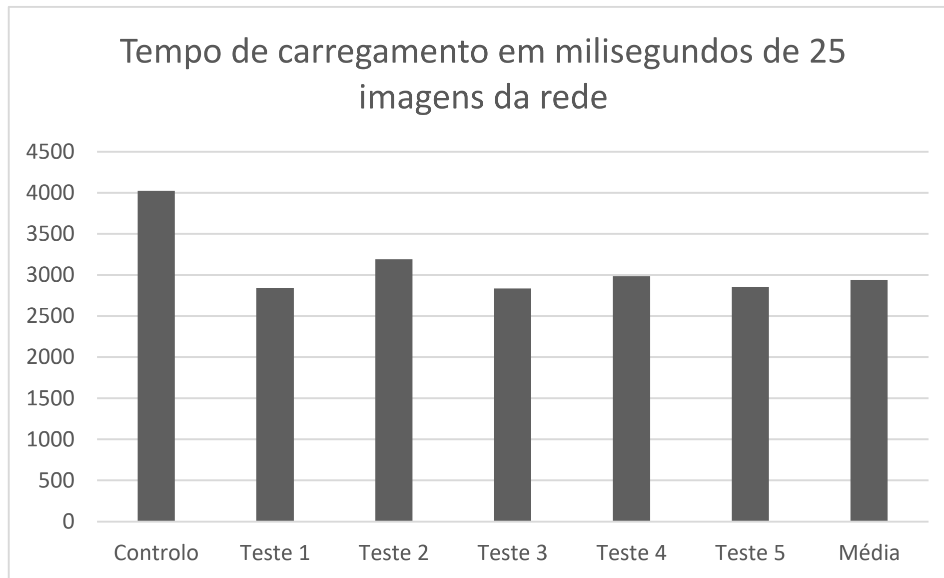



*Figura 37 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (Android)*

Neste caso podemos verificar um tempo de carregamento de cada imagem de aproximadamente 117.62 ms e um desvio de cerca de 5.47 ms correspondente a cerca de 4.65 por cento, obtendo-se uns tempos de carregamento um pouco elevados, mas mesmo assim relativamente aceitáveis uma vez que demora pouco mais de 100 ms no carregamento de cada imagem, continuando a dar uma sensação de ser instantâneo. Podendo-se também esperar tempos de carregamentos homogéneos.

#### 3.5.3.2 PWA

Para o caso da PWA o carregamento das imagens é o normal da framework com um link da mesma para um recurso na internet. E tal como nos testes anteriores foram executadas as

medições dos tempos de carregamento das imagem sendo registados na Tabela 14 e elaborada a representação gráfica presente na Figura 38.

*Tabela 14 - Tempos de carregamento de 25 imagens de endereço da internet (PWA)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Chrome | 516 | 31 | 36 | 39 | 33 | 40 | 35,8 | 3,43 |
| Firefox | 509 | 145 | 133 | 92 | 145 | 98 | 122,6 | 23,04 |
| Opera | 443 | 33 | 37 | 39 | 33 | 30 | 34,4 | 3,20 |

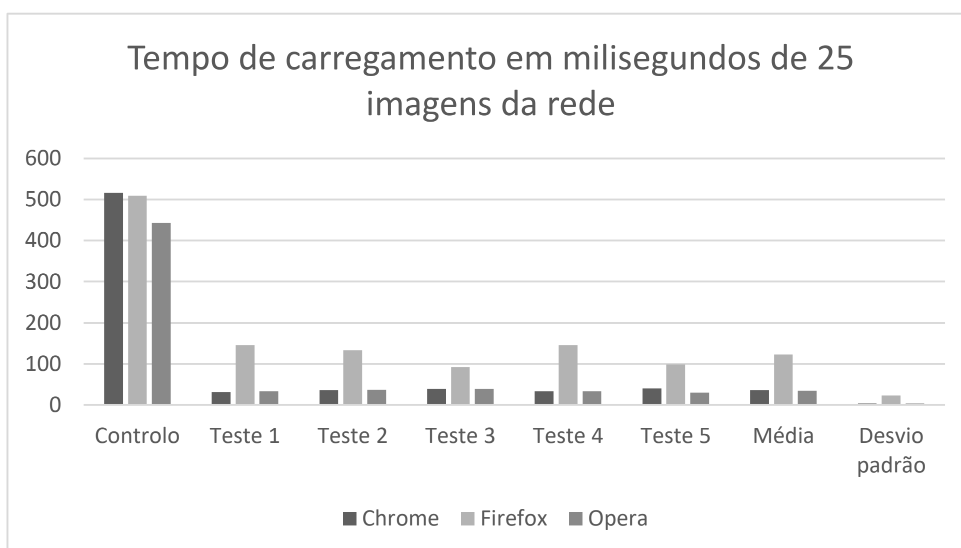



*Figura 38 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (PWA)*

Com uma breve análise a estes resultados podemos verificar que existe uma grande diferença entre os resultados obtidos no Chrome e no Opera em comparação com os obtidos no Firefox. Sendo que também se pode verificar a característica do service worker na utilização da cache, sendo as imagens da pasta static armazenadas em cache, verificamos que o primeiro pedido de controlo é bastante mais alto, sendo os pedidos seguintes muito inferiores, este facto deve-se então a essa propriedade, em que o pedido é efectuado, mas como existe na cache não chega a ser enviado para o exterior e a resposta é dada de imediato pelo servisse worker utilizando os recursos em cache.

Para verificar a existência ou não de diferenças nos tempos em relação a versões instaladas, foram então efectuados também as mesmas medições e registo de tempos para todas as versões instaladas através do browser correspondente dando origem as resultados presentes na Tabela 1 e na Figura 39.

*Tabela 15 - - Tempos de carregamento de 25 imagens de endereço da internet (PWA instalada)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Instalada Chrome | 458 | 32 | 41 | 45 | 36 | 41 | 39 | 4,52 |
| Instalada Firefox | 553 | 189 | 179 | 237 | 197 | 180 | 196,4 | 21,33 |
| Instalada Opera | 417 | 50 | 34 | 40 | 40 | 38 | 40,4 | 5,28 |

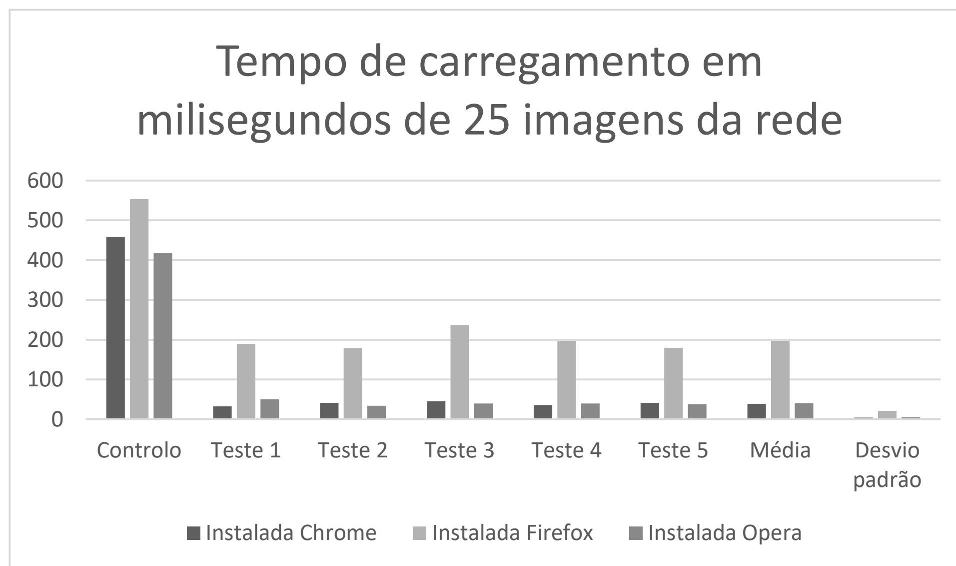



*Figura 39 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (PWA instalada)*

Podendo neste caso também verificar resultados bastante semelhantes quer a nível de tempos de execução como de desvio.

### 3.5.3.3 IONIC

No caso desta framework foi também utilizado o método normal de carregamento de imagem da framework Angular sendo o link da imagem referente a uma imagem alojada num servidor remoto.

*Tabela 16 - Tempos de carregamento de 25 imagens de endereço da internet (IONIC)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 532 | 236 | 249 | 266 | 277 | 253 | 256,2 | 14,1336478 |

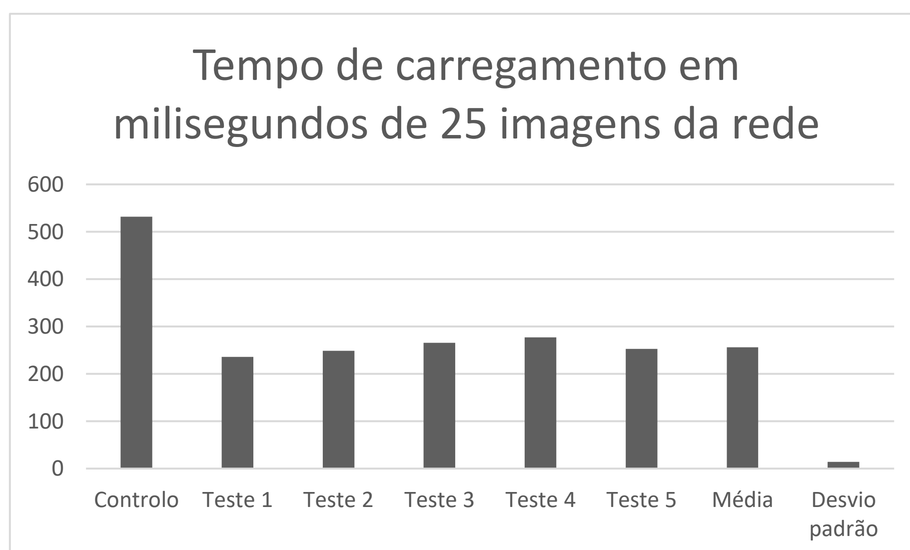



*Figura 40 - Representação gráfica dos resultados obtidos no carregamento de 25 imagens de endereço de rede (IONIC)*

Neste caso podemos verificar um tempo médio de carregamento por imagem de 10.25 ms e um desvio de 0.57 ms correspondente a aproximadamente 5.52 por cento, sendo verificados resultados bastante aceitáveis e bastante homogéneos.

### 3.5.4 Verificação de tempos de execução de tarefa intensiva de processador em Single Thread

Para a verificação de tarefas que requerem a utilização intensiva do processador, foi utilizado o calculo de números primos sendo verificados números de forma sequencial entre zero e cem mil. Sendo verificada a performance tanto a nível da utilização em apenas uma thread como a execução em diversas threads.

###### 3.5.4.1.1 Android

No caso desta tecnologia foi executada de forma normal no código correspondente à thread principal onde corre todo o processamento incluindo o processamento da interface gráfica.

*Tabela 17 - Tempo de execução de tarefa intensiva em single thread (Android)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 2977 | 2200 | 2200 | 2201 | 2200 | 2200 | 2200,2 | 0,4 |

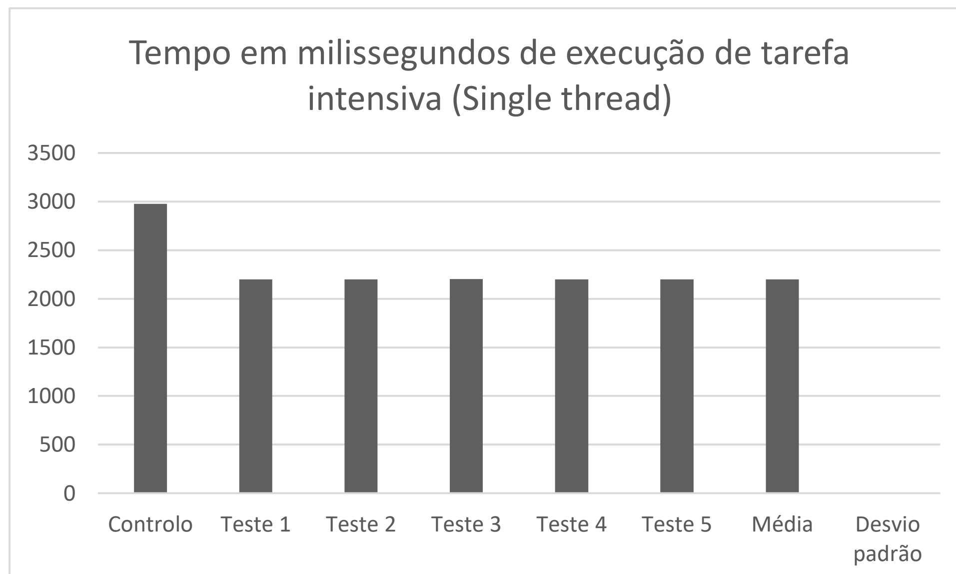



*Figura 41 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (Android)*

Neste teste podemos verificar que as execuções obtiveram tempos médios de 2200 ms sendo o desvio praticamente inexistente.

#### 3.5.4.1.2 PWA

Relativamente à PWA o código foi executado no JavaScript executado no front end sendo executado na inicialização da página para equivaler de certa forma ao comportamento exibido na aplicação Android, sendo efectuada a medição dos tempos totais e efectuado o seu registo como podemos verificar na Tabela 18 e analisar na Figura 42

*Tabela 18 - Tempo de execução de tarefa intensiva em single thread (PWA)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Opera | 3230 | 2796 | 2842 | 2807 | 2800 | 2813 | 2811,6 | 16,28 |
| Firefox | 3972 | 3989 | 3985 | 3955 | 3956 | 3973 | 3971,6 | 14,16 |
| Chrome | 3193 | 2789 | 2799 | 2809 | 2801 | 2804 | 2800,4 | 6,62 |

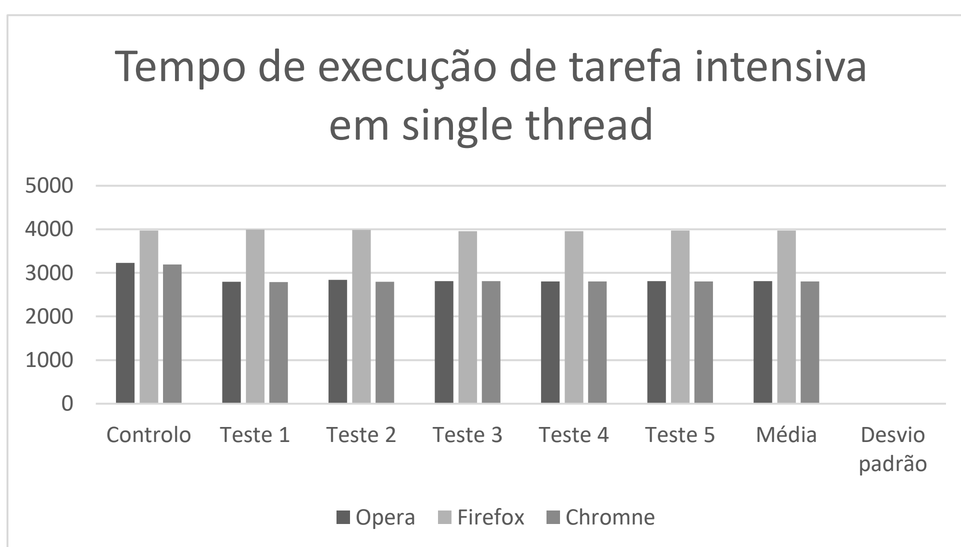



*Figura 42 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (PWA)*

Como anteriormente repetiram-se os testes nos diferentes browsers de forma a poder verificar diferenças entre as duas formas de execução, podendo os resultados ser verificados e analisados na Tabela 19 e na Figura 43.

*Tabela 19 - Tempo de execução de tarefa intensiva em single thread (PWA instalada)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Instalada Chrome | 3207 | 2818 | 2836 | 2799 | 2825 | 2802 | 2816 | 13,93 |
| Instalada Firefox | 3964 | 3978 | 3962 | 3955 | 3942 | 3968 | 3961 | 12,13 |
| Instalada Opera | 3238 | 2791 | 2813 | 2825 | 2823 | 2801 | 2810,6 | 12,99 |

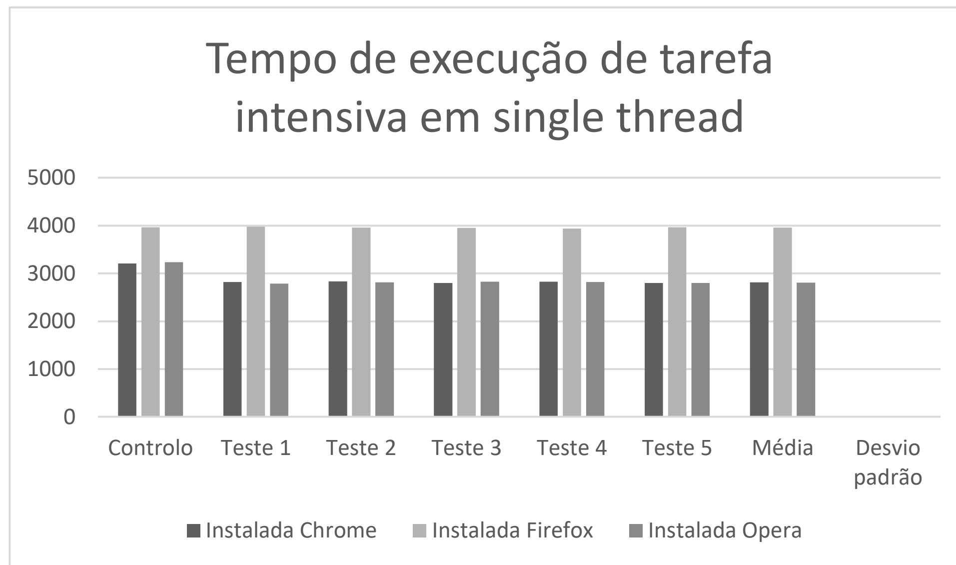



*Figura 43 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (PWA instalada)*

#### 3.5.4.1.3 IONIC

De forma semelhante à PWA o código equivalente em JavaScript foi executado na inicialização da página de forma a simular o mesmo comportamento das aplicações anteriores.

*Tabela 20 - Tempo de execução de tarefa intensiva em single thread (IONIC)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 3174 | 3287 | 3219 | 3198 | 3275 | 3208 | 3237,4 | 36,41208591 |

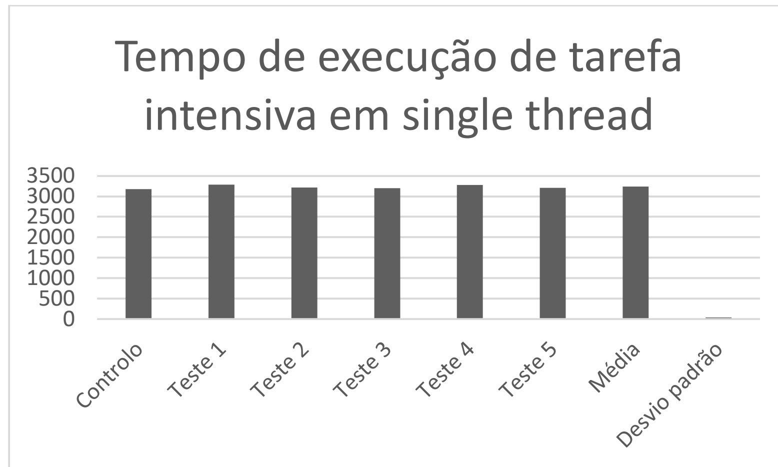



*Figura 44 - Representação gráfica dos resultados obtidos na execução de tarefa intensiva em single thread (IONIC)*

Tendo-se obtido um resultado médio de 3237.4 ms e um desvio quase existente.

### 3.5.5 Verificação de tempos de execução de tarefa intensiva de processador em Multi Thread

#### 3.5.5.1 Android

No caso do multi thread em Android foram exploradas diversas formas de execução do código devido ao seu diferente, foi executado código em threads que ficam a aguardar execução na fila, utilizada uma AsyncTask embora para o processamento de tarefas intensivas como as que se pretende não seja recomendada a sua utilização, querendo demonstrar também a diferença de desempenho da mesma e o porque da não utilização das mesmas. Por ultimo foi também utilizada a framework RxJava, sendo a mesma optimizada para este tipo de execuções e podendo-se escolher o Scheduler, sendo que os mesmos terão diferentes prioridades e a menos que se escolha a Main Thread não irá bloquear a interface gráfica com a execução das tarefas.

#### 3.5.5.1.1 Thread principal

*Tabela 21 - Tempo de execução de 32 tarefas intensiva em multi thread (Android Main Thread)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 71490 | 70439 | 70467 | 70537 | 70433 | 70455 | 70466,2 | 37,36522447 |

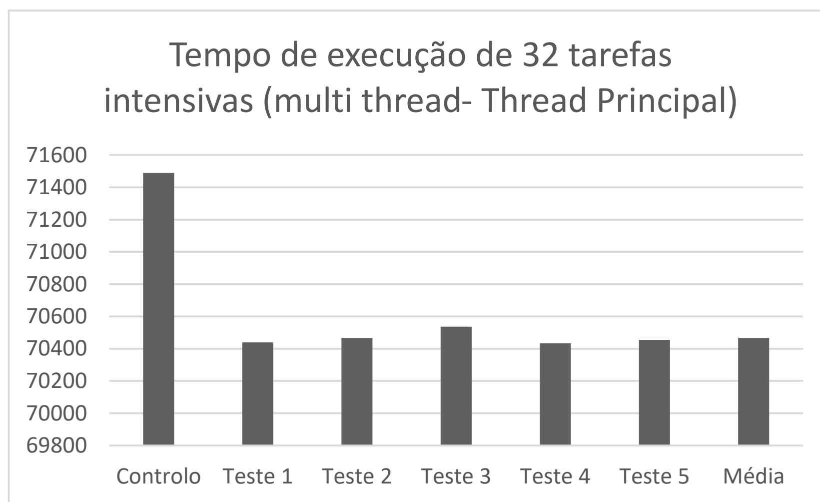



*Figura 45 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (Android Main Thead)*

Com a execução das tarefas nesta Thread e na parte da inicialização, podemos verificar que a interface gráfica apenas é actualizada no final da execução e que o tempo total de execução médio foi de 70466.2 ms correspondendo a um tempo médio de execução de cada tarefa individual de aproximadamente 2202.07 ms sendo relativamente o mesmo tempo de execução de uma tarefa individual repetida 32 vezes.

#### 3.5.5.1.2 AsynckTask

*Tabela 22 - Tempo de execução de 32 tarefas intensiva em multi thread (Android AsyncTask)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 70511 | 70639 | 70528 | 70504 | 70502 | 70567 | 70548 | 51,17421226 |

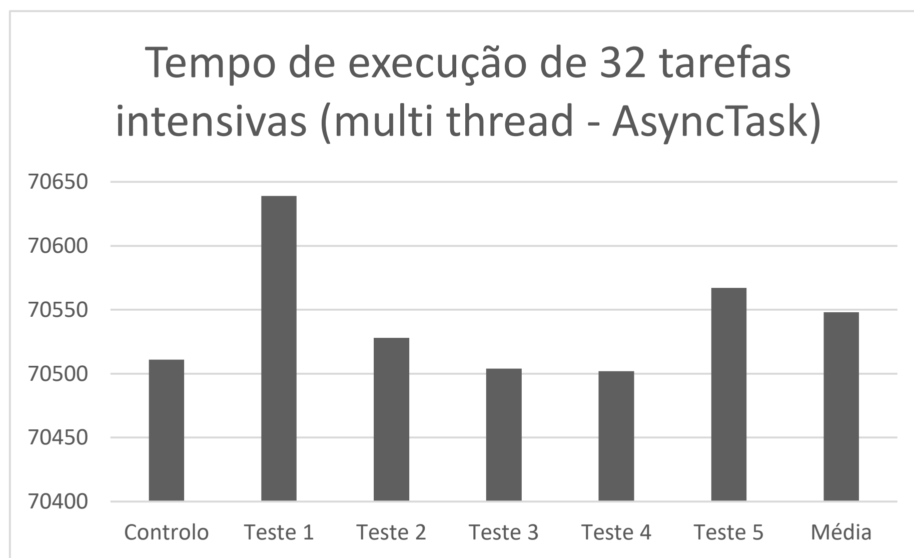



*Figura 46 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (Android AsyncTask)*

Com recurso a AsyncTask podemos verificar que a interface gráfica não bloqueia sendo actualizada consoante os resultados de cada tarefa são obtidos, no entanto verificou-se que os tempos de execução são semelhantes aos anteriores obtendo um tempo total médio de execução de 70548 ms correspondendo a um tempo médio de execução de cada tarefa de 2204.63 ms.

Com recurso ao profiler da ferramenta de desenvolvimento Android Studio, podemos verificar que a execução das tarefas apenas utilizava em média 12 por cento do processador disponível.

#### 3.5.5.1.3 RxJava

*Tabela 23 - Tempo de execução de 32 tarefas intensiva em multi thread (Android RxJava)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 11743 | 11614 | 11591 | 11723 | 11626 | 11638 | 11638,4 | 45,06262309 |

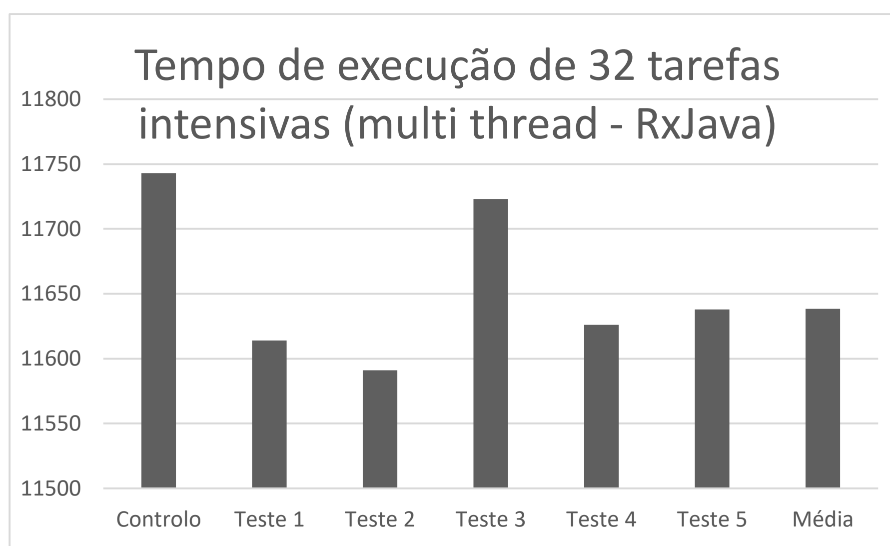



*Figura 47 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (Android RxJava)*

Com recurso ao RxJava e execução das tarefas no scheduler de computação, verificou-se um tempo de execução bastante melhor, sendo que também se verificou uma utilização total do cpu disponível na execução das tarefas, obtendo-se assim um tempo total médio de 11638.4 ms correspondente a um tempo médio por tarefa de aproximadamente 363.7 ms, sendo o melhor resultado e o que será utilizado para comparação final.

### 3.5.5.2 PWA

No caso da PWA e de forma a utilizar recursos do front end, foram utilizados Web Workers, que permitem obter a execução de tarefas em paralelo no JavaScript, para uma utilização mais fácil dos mesmos foi utilizado o módulo VueWorker, sendo este utilizado no código referente à execução no front end. Para a respectiva análise foi efectuada a medição dos tempos de execução da totalidade das 32 tarefas e efectuado o respectivo registo como se pode constar na Tabela 24 e na Figura 48.

*Tabela 24- Tempo de execução de 32 tarefas intensiva em multi thread (PWA)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Chrome | 15705 | 15890 | 15943 | 15692 | 15763 | 15737 | 15805 | 95,32 |
| Firefox | 19109 | 19103 | 19056 | 18957 | 18997 | 18899 | 19002,4 | 71,80 |
| Opera | 15516 | 15600 | 15507 | 15488 | 15602 | 15596 | 15558,6 | 50,29 |

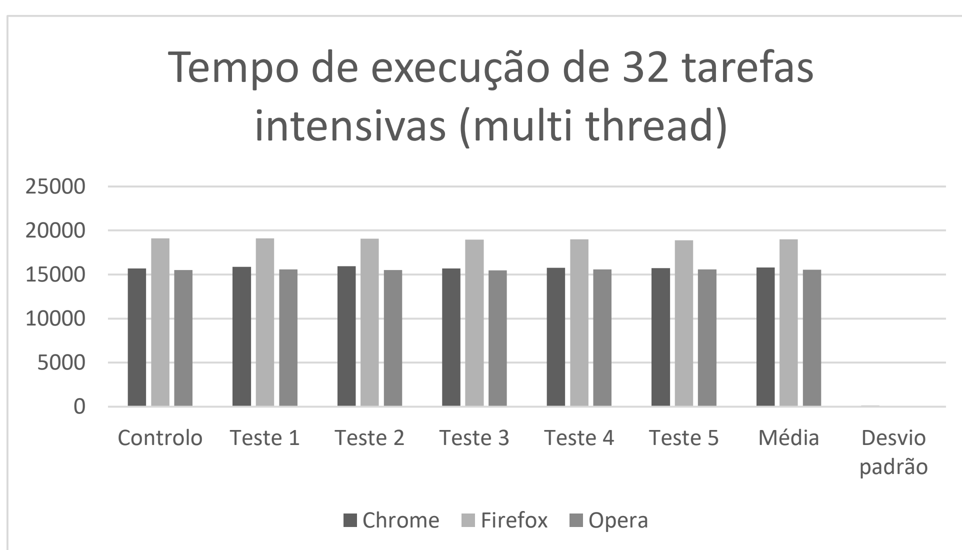



*Figura 48 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (PWA)*

Numa breve análise destes resultados podemos verificar uma consistência dos mesmos bem como a obtenção de resultados bastantes semelhantes em relação às execuções no Chrome e Opera sendo que o Firefox obteve resultados com valores relativamente superiores. Para

verificação de resultados entre versões foram então realizados todos os mesmos testes nas versões instaladas a partir dos respectivos browsers, podendo-se verificar os resultados na Tabela 25 e na Figura 49.

*Tabela 25 - Tempo de execução de 32 tarefas intensiva em multi thread (PWA instalada)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Instalada Chrome | 15714 | 15823 | 15911 | 15824 | 15846 | 15887 | 15858,2 | 35,14 |
| Instalada Firefox | 18931 | 18877 | 18954 | 19059 | 18910 | 18966 | 18953,2 | 61,69 |
| Instalada Opera | 15450 | 15731 | 15843 | 15675 | 15616 | 15591 | 15691,2 | 90,09 |

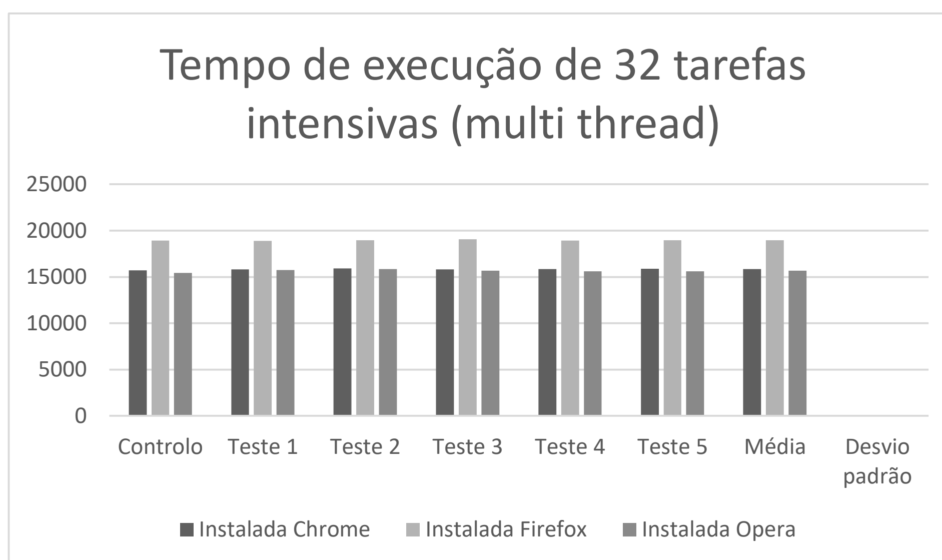



*Figura 49 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (PWA instalada)*

Numa breve análise aos mesmos podemos verificar que existe apenas uma muito ligeira vereação, mantendo-se todos os restantes aspectos.

#### 3.5.5.3 IONIC

Nesta tecnologia para obter o mesmo efeito, foi utilizado o módulo ngx-web-worker, que permite obter os mesmos resultados que o anterior módulo utilizado na PWA cujo intuito era ser utilizado com a framework VueJs.

*Tabela 26 - Tempo de execução de 32 tarefas intensiva em multi thread (IONIC)*

| 32 | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 15881 | 15834 | 15753 | 15950 | 16106 | 15780 | 15884,6 | 129,7036622 |

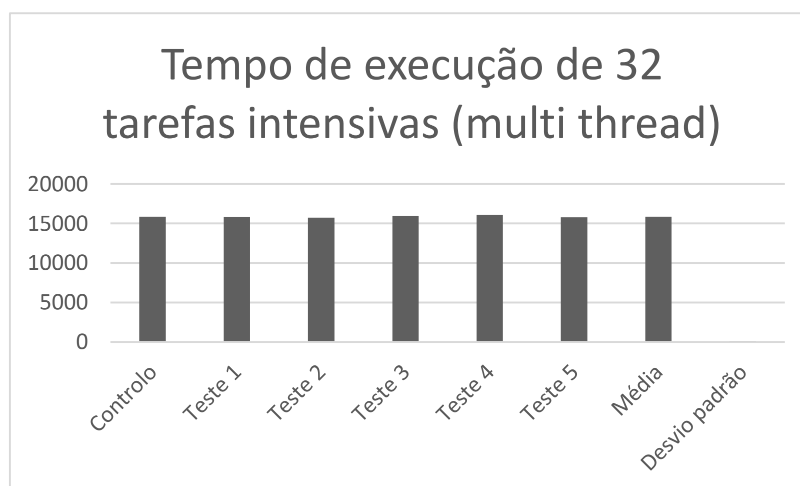



*Figura 50 - Representação gráfica dos resultados obtidos na execução de 32 tarefas intensivas em multi thread (IONIC)*

Apesar da diferença das frameworks utilizadas continuou a verificar-se pedidos bastante homogéneos, tendo se registado um valor médio para o tempo total de execução de 15884.6 ms correspondendo a aproximadamente 496.39 ms para cada execução individual.

### 3.5.6 Operações de rede

Para a determinação da performance de pedidos de rede foram utilizados os mesmos métodos de realização de pedidos que foram utilizados nos testes de login e registo, sendo que desta vez irá ser apenas efectuado o login uma vez para aquisição de um token de autenticação, sendo de seguida realizados os pedidos da listagem de boxes 70 vezes e para cada um desses pedidos irá ser efectuado um número de peidos igual ao total de utilizadores registados em cada uma das VITABOXES listadas.

#### 3.5.6.1 Android

Foi efectuado um pedido inicial para obtenção do referido token, sendo a sequencia de pedidos accionada no listener correspondente ao pedido inicial, os pedidos de listagem de utilizadores são do mesmo modo, efectuados no listener do pedido de listagem das VITABOXES, sendo o tempo total registado a quando da recepção do ultimo pedido da listagem de utilizadores.

*Tabela 27 – Registo de tempos de execução de pedidos de rede interligados (Android)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 5705 | 4894 | 4613 | 4975 | 5781 | 5179 | 5088,4 | 391,0394354 |

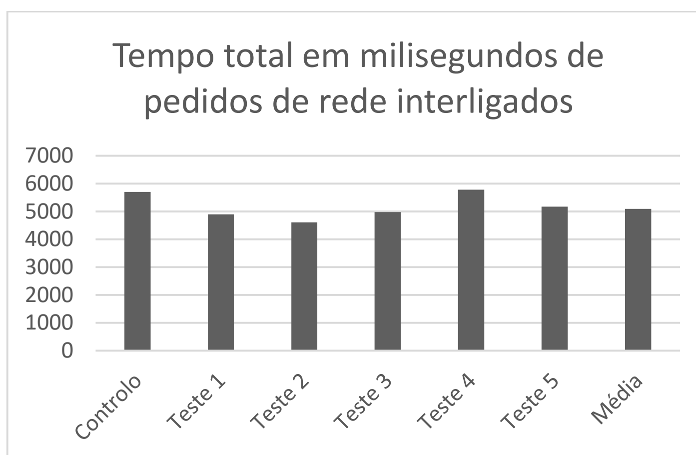



*Figura 51 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (Android)*

Com os testes realizados podemos constatar um tempo médio total de execução de 5088.4 ms e um desvio padrão que ronda os 7.69 por cento, sendo expectáveis tempos relativamente semelhantes ao mencionado em execuções similares.

### 3.5.6.2 PWA

Também neste caso foi utilizado o mesmo modo de efectuar pedidos e ligação entre pedidos, a diferença apenas que em vez os pedidos serem desencadeados nos listeners, são desencadeados nos retornos das promessas. Foi então realizada a medição dos tempos totais de execução e respectivo registo na Tabela 28 podendo estes ser analisados na sua respectiva representação gráfica na Figura 52.

*Tabela 28 - Registo de tempos de execução de pedidos de rede interligados (PWA)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Chrome | 13874 | 16005 | 13406 | 12591 | 11043 | 13719 | 13352,8 | 1617,81413 |
| Firefox | 12618 | 12806 | 11705 | 8836 | 9375 | 12474 | 11039,2 | 1627,7085 |
| Opera | 24954 | 12472 | 13537 | 10761 | 10358 | 12039 | 11833,4 | 1155,783821 |

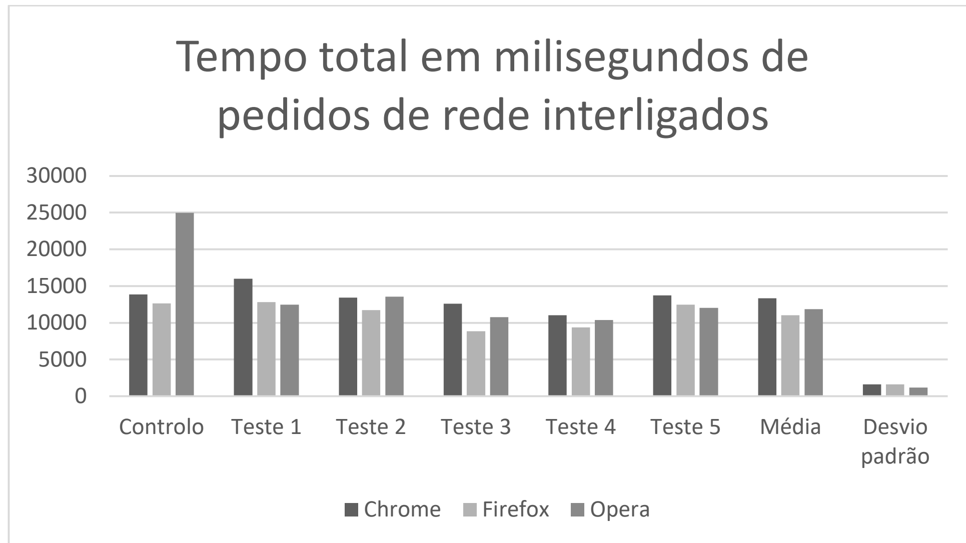



*Figura 52 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (PWA)*

Efectuando uma breve análise aos resultados podemos verificar que neste caso o browser Firefox obteve os melhores resultados, tendo opera resultados bastante próximos e Chrome os piores resultados dos três. Podemos também verificar que existe alguma irregularidade nos resultados obtidos, não podendo ser expectável um resultado uniforme na execução deste tipo de operações. De forma a novamente se poder determinar se existem discrepâncias entre a versão instalada e a correr no browser, foram então repetidos os testes desta vez nas versões instaladas a partir do browser correspondente, podendo os resultados ser verificados e analisados na Tabela 29 e na Figura 53.

*Tabela 29 - Registo de tempos de execução de pedidos de rede interligados (PWA instalada)*

| Tempo ms | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Instalada Chrome | 13562 | 14217 | 14730 | 12910 | 13810 | 10377 | 13208,8 | 1536,25 |
| Instalada Firefox | 17449 | 13983 | 9531 | 13313 | 11641 | 12653 | 12224,2 | 1552,82 |
| Instalada Opera | 10655 | 10551 | 12112 | 14695 | 12497 | 12320 | 12435 | 1325,02 |

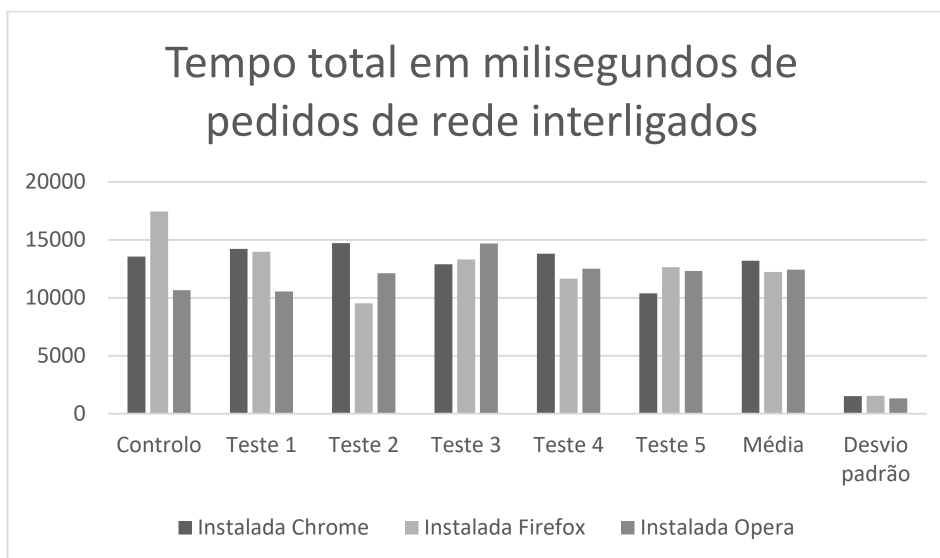



*Figura 53 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (PWA instalada)*

Com uma breve análise podemos verificar que os resultados se mantem embora exista uma ligeira variação dos tempos totais necessários à realização das operações.

#### 3.5.6.3 IONIC

Também nesta framework foi mantido o mesmo método de efectuar pedidos utilizados para o registo e login, sendo então os pedidos tal como no caso da framework anterior, interligados nos resultados das promessas.

*Tabela 30 - Registo de tempos de execução de pedidos de rede interligados (IONIC)*

| | Controlo | Teste 1 | Teste 2 | Teste 3 | Teste 4 | Teste 5 | Média | Desvio padrão |
|---|---|---|---|---|---|---|---|---|
| Tempo em milissegundos | 11922 | 10626 | 13050 | 12854 | 15253 | 14170 | 13190,6 | 1544,212369 |

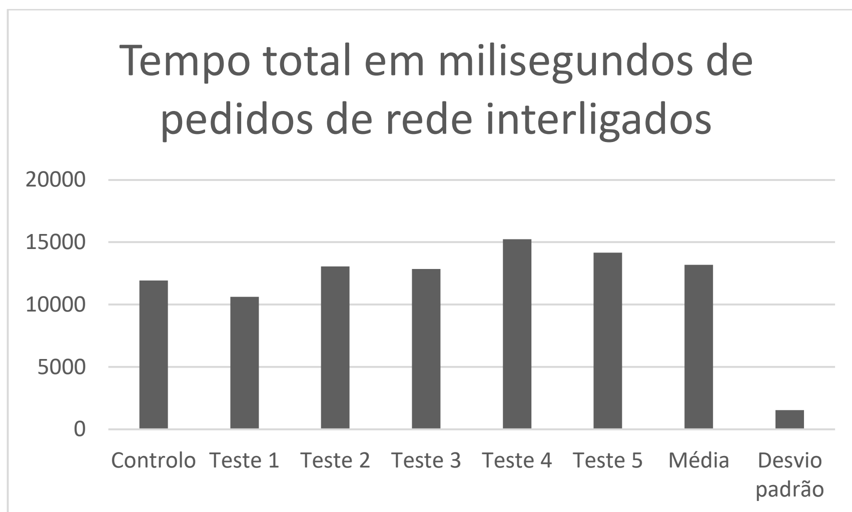



*Figura 54 - Representação gráfica dos resultados obtidos na execução de pedidos de rede interligados (IONIC)*

Apresentando, no entanto, resultados um pouco dispersos com valor total médio de 13190.6 ms e um desvio de aproximadamente 1544.21 ms correspondente a 11.71 por cento, existindo uma diferença relativa entre todos os pedidos da série executada.

# 4 Gestão de memória

Para verificar o consumo e método de gestão de memória foi realizado um teste em que se procede à abertura consecutiva do mesmo ecrã por o número de repetições indicadas, sendo utilizado a ferramenta Profiler do Android Studio que monitoriza entre outras, o consumo de memória instantâneo.

## 4.1 Android

Neste teste podemos verificar como é o procedimento normal desta plataforma, um crescente aumento do consumo da memória disponível, sendo activado o mecanismo de garbage colector quando o dispositivo precisa de libertar memória ou existem referencias que se encaram como podendo ser descartadas, como se pode verificar na Figura 55.

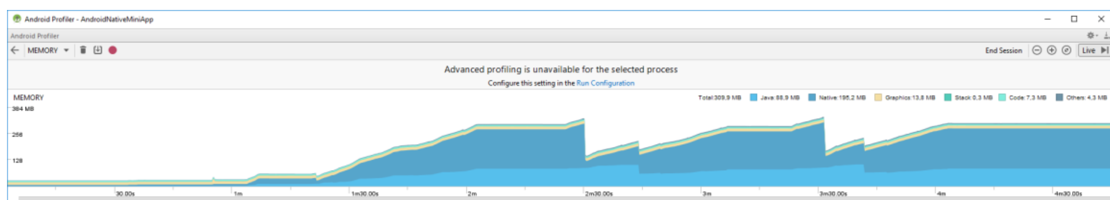



*Figura 55 – Gestão de memória Android*

## 4.2 PWA

No caso das PWA's a situação torna-se um pouco diferente, apesar de não existir o mecanismo de gestão de memória, a tarefa de abertura de uma nova actividade, corresponde neste caso apenas a navegação para outra rota, sendo que para alem desse facto a framework irá reutilizar os componentes existente, criando novas referencias apenas caso os componentes não existam actualmente. Devido a ambos os factores mencionados

anteriormente, podemos observar um consumo de memória relativamente constante, não sendo observadas mudanças significativas no consumo de memória, sendo também este relativamente baixo como se pode verificar na Figura 56.

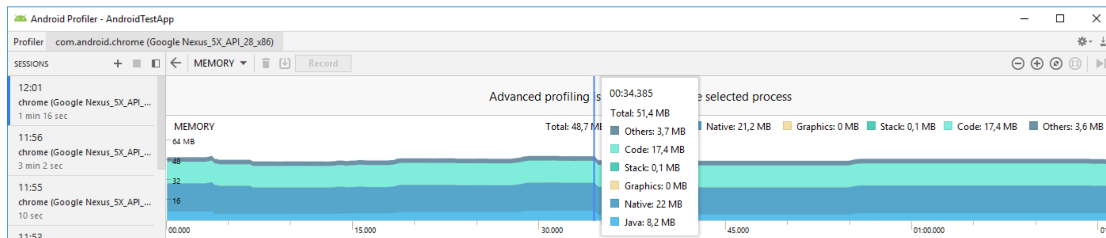



*Figura 56 – Gestão de memória PWA*

## 4.3 IONIC

Neste ultimo caso podemos verificar que apesar do código gerado ser executado numa webview, devido ao facto de ser uma aplicação e como tal ter o seu necessário alocação de memória, existe então a webview sendo novamente alocada memória para esse componente e o código a este associado, como tal verifica-se uma alocação total de memória superior. No entanto após esta alocação mais elevada, o mecanismo de reutilização e navegação é igual ao mencionado nas PWA sendo expectável um consumo de memória relativamente constante como se pode verificar na Figura 57.

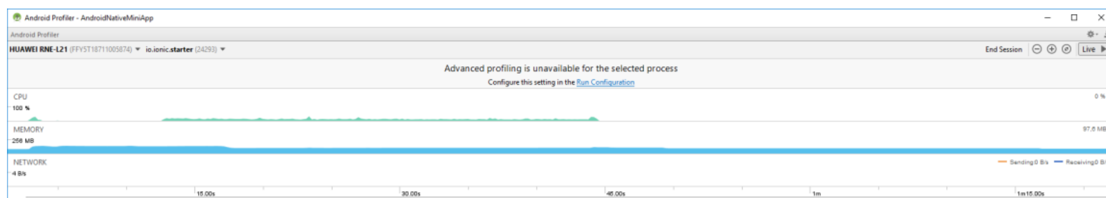



*Figura 57 – Gestão de memória IONIC*

# 5 Usabilidade

Um outro teste que se decidiu realizar foram testes de usabilidade, como tal foram desenvolvidas duas mini-aplicações representativas de um exemplo simples de front end, de forma a que se pudesse verificar a forma de utilização tanto nativa como de uma das frameworks seleccionadas, neste caso o conceito de PWA com utilização de VueJS, que foi seleccionada com base na avaliação dos resultados obtidos e explorados nos pontos anteriores.

Com o objectivo de proporcionar uma experiencia o mais nativa possível, foi elaborada primeiramente a aplicação em Android nativo, procedendo-se de seguida a realização da PWA tentando aproximar o funcionamento o máximo possível aquele que a aplicação Android apresentava.

Para ambas as aplicações as funcionalidades que se escolheu tentar implementar foram:

- Login
- Registo
- Listagem de VITABOXES
- Listagem de pacientes
- Listagem de boards
- Listagem de Sponsors
- Registo de Vitaboxes

Foi efectuado o desenvolvimento das funcionalidades mencionadas em Android nativo, no decorrer do desenvolvimento da PWA e devido a uma necessidade de gestão de tempo, tendo em conta que as funcionalidades de listagem seriam de apresentação semelhante, a PWA foi apenas parcialmente desenvolvida, sendo que se pode obter uma comparação de usabilidade com as funcionalidades implementadas na mesma.

Com uma interface em tudo semelhante ou quase igual, sendo que foi utilizada como base do design as guidelines da Google , material design, e com a existência de uma framework para VueJs com componentes desenvolvidos de forma a respeitar essas mesmas guidelines, pode ser desenvolvido um interface gráfico em que se verificou uma usabilidade igual em

ambos os casos, sendo possível replicar os comportamentos e elementos de navegação da plataforma nativa.

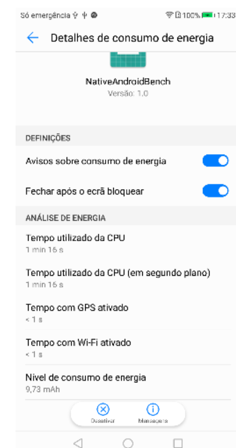
Após esta análise pode-se constatar que a usabilidade será possível de implementar de forma igual em ambas as plataformas sendo alguma possível complicação de implementação apenas influenciada por implementações de design.

# 6 Consumo de bateria máximo estimado

Para que se obtivesse um consumo de energia estimado a máximo ou próximo de máximo foi efectuada a medição do consumo a quando da execução da tarefa intensiva em multi thread finalizando a medição assim que a tarefa era terminada.

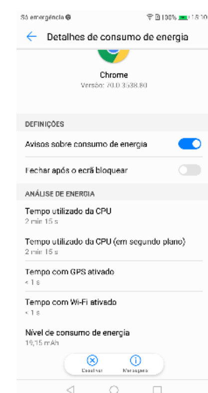
Foi também calculado o valor por segundo de forma a pode efectuar uma comparação total de consumo entre tecnologias/frameworks

## 6.1 Android




Com um consumo registado de 9.73 mAh numa duração de 1 minuto e 16 segundos verificando-se um consumo por segundo de aproximadamente 0.13 mAh.

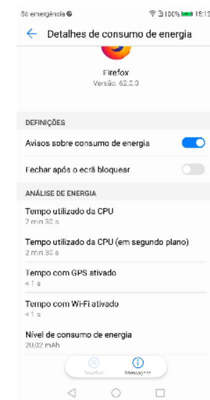
## 6.2 PWA



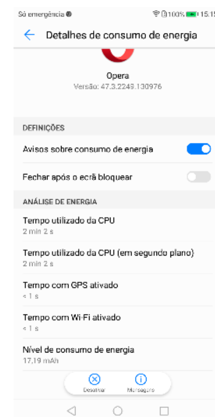

Ao correr as respectivas versões instaladas verificou-se que apenas era registada a utilização do browser correspondente, sendo deste modo apresentados apenas os resultados para os browsers correspondentes.

Relativamente à PWA executada no browser Chrome foi registado um consumo total de 19.15 mAh para um total de 2 minutos e 15 segundos, verificou-se um registo por segundo de aproximadamente 0.14 mAh

Relativamente à PWA executada no browser Firefox foi registado um consumo total de 20.02 mAh para um total de 2 minutos e 30 segundos, verificou-se um registo por segundo de aproximadamente 0.13 mAh

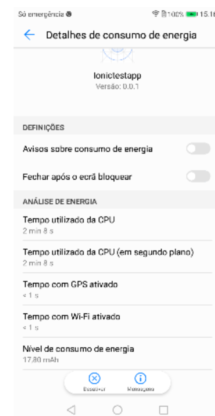



Relativamente à PWA executada no browser Firefox foi registado um consumo total de 17.19 mAh para um total de 2 minutos e 2 segundos, verificou-se um registo por segundo de aproximadamente 0.14 mAh




## 6.3 IONIC

Relativamente IONIC foi registado um consumo total de 17.19 mAh para um total de 2 minutos e 2 segundos, verificou-se um registo por segundo de aproximadamente 0.14 mAh




Com estes resultados podemos verificar que os consumos são semelhantes embora nativo e PWA Firefox apresentem resultados ligeiramente melhores com 0.01mAh por segundo a menos.

# 7 Simplicidade de código e facilidade de manutenção

Dois outros pontos que se decidiu analisar foi a simplicidade do código para alcançar o mesmo objectivo e a facilidade de manutenção da aplicação.

## 7.1 Simplicidade do código

Relativamente à simplicidade do código, foram escolhidas duas tarefas em que as mesmas fossem executadas de formas distintas de acordo com a tecnologia/framework em que se está a desenvolver. Para esse efeito foram identificados os casos da interligação de pedidos de rede e processamento de tarefas com utilização intensiva do CPU, sendo que no caso do código a ser comparado no Android dos métodos implementados será o caso de RxJava uma vez que é o método que proporciona resultados semelhantes.

Relativamente à interligação de pedidos de rede, verificou-se que nos casos da PWA e IONIC, a simplicidade é semelhante, a base do código de interligação em tem por bases os mesmos pontos de repetição e novos pedidos nas promessas. Já no caso de Android, mesmo utilizando uma framework para realização dos pedidos, no caso retrofit, o código apesar de simples de entender e seguir, é mais extenso e implica a existência de mais código separado e um pouco repetitivo, embora necessário. Exemplos dos códigos podem ser verificados na Figura 58,Figura 59 e Figura 60.

```
methods: {
  startProcess () {
    this._data.status = 'Processing, please wait'
    var start = Date.now()
    this._data.requestMade = 0      You, a few seconds ago • Uncommitted changes
    var data = { email: this.email, password: this.password }
    var ctx = this
    this.makePostRequest('/login', data, ctx._data.settings.config, function (response) {
      ctx._data.settings.config.headers.Authorization = response.data.token
      for (var i = 0; i < ctx._data.numberOfTimes; i++) {
        ctx.makeGetRequest('/vitabox', ctx._data.settings.config, function (response) {
          ctx._data.requestsToMake += response.data.vitaboxes.length
          response.data.vitaboxes.forEach(vitabox => {
            ctx.makeGetRequest('/vitabox/' + vitabox.id + '/patient', ctx._data.settings.config, function () {
              ctx._data.requestMade += 1
              if (ctx._data.requestsToMake === ctx._data.requestMade) {
                ctx._data.status = 'Task finished in: ' + Math.ceil(Date.now() - start) + 'ms'
              }
            })
          })
        })
      }
    })
  },
```

*Figura 58 – Interligação de peidos de rede PWA*

```
32    ngOnInit() {}
33
34    callAPI() {
35      var start = Date.now();
36      this.data.email = this.email;
37      this.data.password = this.password;
38      this.requestsMade = 0;
39      this.requestsToMake = 0;
40      this.status = "Processing please wait";
41      let options = new RequestOptions({ headers: new Headers(this.headers) });
42      this.restService.doPost("/login", this.data, options).then(response => {
43        this.headers.Authorization = response["token"];
44        options = new RequestOptions({ headers: new Headers(this.headers) });
45        for (var i = 0; i < this.numberOfTimes; i++) {
46          this.restService.doGet("/vitabox", options).then(boxList => {
47            this.requestsToMake += boxList['vitaboxes'].length
48            boxList["vitaboxes"].forEach(box => {
49              this.restService
50                .doGet("/vitabox/" + box.id + "/patient", options)
51                .then(patients => {
52                  this.requestsMade++;
53                  if (this.requestsMade === this.requestsToMake) {
54                    this.status = "Process finished in " + (Date.now() - start) + "ms";
55                  }
56                });
57            });
58          });
59        }
60      });
61    }
62  }
63
```

*Figura 59 – Interligação de pedidos IONIC*

```
private void makeRequests() {
    requestCounter = 0;
    String number = numberOfRequests.getText().toString();
    requestsToMake = Integer.parseInt(number);
    startTime = Calendar.getInstance();
    callLogin();
}

//step1 login to get token
private void callLogin() {
    ApiCallInterface apiService = ServiceCallManager.getServiceCallManager(AppConfig.getServerAddress(),
            timeout: null, getApplicationContext()).create(ApiCallInterface.class);
    Call<LogInResponse> callLogin = apiService.login(new LogInRequestBody(email.getText().toString(), password.getText().toString()));

    callLogin.clone().enqueue(getLoginCallback);
}

private Callback<LogInResponse> getLoginCallback = new Callback<LogInResponse>() {
    @Override
    public void onResponse(final Call<LogInResponse> call, Response<LogInResponse> response) {
        if (response.body() != null) {
            AppConfig.setToken(response.body().getToken());
            for (int i = 1; i <= requestsToMake; i++) {
                callListBoxs();
            }
        }
    }

    @Override
    public void onFailure(Call<LogInResponse> call, Throwable t) {
    }
};

//step2 get all boxes
private void callListBoxs() {
    ApiCallInterface apiService = ServiceCallManager.getServiceCallManager(AppConfig.getServerAddress(),
            timeout: null, getApplicationContext()).create(ApiCallInterface.class);
    Call<ListBoxsResponse> callRegister = apiService.listBoxs();

    callRegister.clone().enqueue(getListBoxsCallback);
}

private Callback<ListBoxsResponse> getListBoxsCallback = new Callback<ListBoxsResponse>() {
    @Override
    public void onResponse(final Call<ListBoxsResponse> call, Response<ListBoxsResponse> response) {
        if (response.body() != null) {
            List<Vitabox> boxes = response.body().getVitaboxes();
            numberOfPatientsRequests += boxes.size();
            for (int i = 0; i < boxes.size(); i++) {
Networking
```

*Figura 60 – Interligação de pedidos Android*

Em termos da execução das tarefas de utilização intensiva do processador, começando com o código referente à PWA, encontrou-se um problema na execução do código referente ao calculo dos números primos, não sendo possível referenciar uma função declarada ao mesmo nível da função em que se encontra a chamada da criação da thread, tendo sido executadas algumas tentativas, não se conseguindo de outra forma que a duplicação do código, talvez carecendo uma investigação mais profunda, tentando identificar se o problema é do código efectuado, contexto ou mesmo do módulo utilizado. Relativamente ao código elaborado na construção da aplicação em IONIC, foi possível utilizar a função evitando repetições de código, sendo o restante código bastante similar ou mesmo igual. Por ultimo na aplicação Android nativo, com a utilização da framework de RxJava foi possível elaborar código não muito extenso e relativamente simples, sendo apenas o mais complicado os conceitos inerentes à ao modo de funcionamento da própria framework. Podendo-se verificar exemplos dos códigos mencionados na Figura 61,Figura 62 e Figura 63.

```
17  mounted: function () {
18    var count = 0
19    var startTime = Date.now()
20    if (this.$route.params.mode === '0') {
21      this.cpuIntensiveTask()
22    } else if (this.$route.params.mode === '1') {
23      for (var i = 1; i <= 32; i++) {
24        this.$worker.run((args) => {
25          for (var x = 0; x <= 100000; x++) {
26            for (var y = 2; y < x; y++) {
27              if (x % y === 0) {
28                // console.log(x + ' Não é primo.')
29                break
30              }
31            }
32            // console.log(x + ' é primo.')
33          }
34          return args
35        }, [i])
36          .then(result => {
37            // console.log('Task finished: ' + result)
38            this._data.results.push('Task finished: ' + result)
39            count++
40            if (count === 32) {
41              // console.log('Process finished in: ' + Math.ceil(Date.now() - startTime) + 'ms')
42              this._data.results.push('Process finished in: ' + Math.ceil(Date.now() - startTime) + 'ms')
43            }
44          })
45          .catch(e => {
46            console.error(e)
47          })
48      }
49    }
50  },
51  methods: {
52    cpuIntensiveTask () {
53      var start = Date.now()
54      for (var x = 0; x < 100000; x++) {
55        for (var y = 2; y < x; y++) {
56          if (x % y === 0) {
57            // console.log(x + 'Não é primo.')
58            break
59          }
60        }
61        // console.log(x + ' é primo.')
62      }
63      // console.log('Task finished in: ' + Math.ceil(Date.now() - start) + 'ms')
64      this._data.results.push('Task finished in: ' + Math.ceil(Date.now() - start) + 'ms')
```

*Figura 61 – Exemplo de código tarefa de utilização intensiva do processador (PWA)*

```
22
23  ngOnInit() {
24    var count = 0;
25    var startTime = Date.now();
26    if (this.single === "0") {
27      this.results.push(this.cpuIntensiveTask(undefined));
28      this.results.push(
29        "Process finished in: " + Math.ceil(Date.now() - startTime) + "ms"
30      );
31    } else if (this.single === "1") {
32      for (var i = 1; i <= 32; i++) {
33        const promise = this._webWorkerService.run(this.cpuIntensiveTask,i);
34        promise.then(result => {
35          this.results.push(result);
36          count++;
37          if (count === 32) {
38            // console.log('Process finished in: ' + Math.ceil(Date.now() - startTime) + 'ms')
39            this.results.push(
40              "Process finished in: " + Math.ceil(Date.now() - startTime) + "ms"
41            );
42          }
43        });
44      }
45    }
46  }
47
48  cpuIntensiveTask(val) {        Ricardo Anacleto, 3 months ago • cpuIntensiveTasks
49    var start = Date.now();
50    for (var x = 0; x < 100000; x++) {
51      for (var y = 2; y < x; y++) {
52        if (x % y === 0) {
53          // console.log(x + 'Não é primo.')
54          break;
55        }
56      }
57      // console.log(x + ' é primo.')
58    }
59    return "Task " + (val != undefined ? val.toString() : "" ) + " finished in: " + Math.ceil(Date.now() - start) + "ms";
60  }
61 }
62
```

*Figura 62 - Exemplo de código tarefa de utilização intensiva do processador (IONIC)*

```
35          for (int i = 1; i <= totalRequests; i++) {
36              onRunCpuIntensiveTask(i);
37          }
38      }
39
40      private Integer cpuIntensiveTask(int i) {
41          for (int x = 0; x < 100000; x++) {
42              for (int y = 2; y < x; y++) {
43                  if (x % y == 0) {
44                      break;
45                  }
46              }
47          }
48          return i;
49      }
50
51      void onRunCpuIntensiveTask(int i) {
52          disposables.add(getCpuIntensiveTaskObservable(i)
53                  // Run on a background thread
54                  .subscribeOn(Schedulers.computation())
55                  // Be notified on the main thread
56                  .observeOn(AndroidSchedulers.mainThread())
57                  .subscribeWith(new DisposableObserver<Integer>() {
58                      @Override
59                      public void onComplete() {
60                      }
61
62                      @Override
63                      public void onError(Throwable e) {
64                      }
65
66                      @Override
67                      public void onNext(Integer integer) {
68                          count++;
69                          TextView text = new TextView(getApplicationContext());
70                          text.setText("Task finished: " + integer);
71                          resultsContainer.addView(text);
72                          if (count == totalRequests) {
73                              text = new TextView(getApplicationContext());
74                              text.setText("Process finished in: " + (Calendar.getInstance().getTimeInMillis() - startTime.getTimeInMillis()));
75                              resultsContainer.addView(text);
76                          }
77                      }
78                  }));
79      }
80
81      public Observable<Integer> getCpuIntensiveTaskObservable(final int i) {
82          return Observable.defer((Callable) () → {
85                  return Observable.just(cpuIntensiveTask(i));
```

*Figura 63 - Exemplo de código tarefa de utilização intensiva do processador (Android)*

Com a análise destes factos e excertos de código podemos verificar que tanto a framework VueJs e AngularJS utilizadas no desenvolvimento das aplicações PWA e IONIC respectivamente, permitem elaborar código mais simplificado que o apresentado na aplicação nativa de Android.

## 7.2 Facilidade de manutenção do código

Relativamente à facilidade de manutenção do código, podemos verificar que a simplicidade é um factor que privilegia tanto a PWA e IONIC, sendo que no caso da PWA mais especificamente VueJS a utilização do código num único componente poderá ser encarada como uma forma mais simples do que a separação em vários ficheiros relacionados com o mesmo ecrã existente em AngularJs, sendo apenas uma questão de preferências pessoais de cada programador, não é considerada uma desvantagem pois permite uma maior separação e como tal facilidade em saber onde encontrar cada código para o devido efeito. Android, no entanto, alem de apresentar por vezes uma maior complexidade de código como já foi mencionado, tem como desvantagem o facto de ser apenas referente a uma plataforma, sendo que no caso das outras tecnologias/frameworks abordadas, apresentar a capacidade de execução em diversas plataformas com o mesmo código.

Tendo em conta os factos mencionados determina-se que será mais fácil de mantar o código quer numa PWA como uma aplicação desenvolvida com recurso a IONIC.

# 8 Resultados

Após a realização de todos os testes e respectiva análise sumária, procedeu-se à aglomeração dos resultados através dos seus valores médios para tempos totais de execução das tarefas, procedendo à identificação da tecnologia ou framework que obteve o melhor e pior resultado de cada tipo de teste efectuado como demonstrado na Tabela 31.

*Tabela 31 – Comparação geral dos tempos obtidos*

| | Android | PWA- Ins. Chrome | PWA Chrome | PWA - Ins. Firefox | PWA - FireFox | PWA Instalada Opera | PWA - Opera | IONIC |
|---|---|---|---|---|---|---|---|---|
| **Login** | 2264,8 | 1606,6 | 1708,8 | 2074,8 | 1877 | 1849,2 | 1671,2 | 1583,4 |
| **Registar** | 1066,2 | 1024,4 | 999,6 | 1154 | 1012,8 | 1122 | 1016,2 | 1043,4 |
| **Imagem do disco** | 67,2 | 100,4 | 102,2 | 347,2 | 309,2 | 97 | 105,6 | 680,8 |
| **Imagem da rede** | 2940,4 | 39 | 35,8 | 196,4 | 122,6 | 40,4 | 34,4 | 256,2 |
| **CPU ST** | 2200,2 | 2816 | 2800,4 | 3961 | 3971,6 | 2810,6 | 2811,6 | 3237,4 |
| **CPU MT** | 11638,4 | 15858,2 | 15805 | 18953,2 | 19002,4 | 15691,2 | 15558,6 | 15884,6 |
| **Pedidos de rede** | 5088,4 | 13208,8 | 12224,2 | 12224,2 | 11039,2 | 12435 | 11833,4 | 13190,6 |

Com uma breve análise e tendo em conta apenas a performance obtida, podemos verificar que Android nativo na sua maioria obteve os melhores resultados, tendo, no entanto, obtido também o prior resultado em dois dos testes, sendo a diferença bastante significativa nesses mesmos testes, mesmo assim é detentor do maior número de testes com melhor resultado sendo na sua generalidade a melhor escolha a nível de performance global dos testes realizados.

No entanto e de forma a se ter uma melhor percepção dos resultados entre tecnologias/framework, foi decidido realizar uma "filtragem" aos mesmos, começando pela selecção do melhor conjunto de testes das PWA's, procedendo-se aglomeração desses mesmos resultados numa tabela a parte, identificando não só os melhores e piores resultados como anteriormente como também o segundo pior resultado de forma a se obter uma melhor percepção visual dos mesmos como se verifica na Tabela 32.

*Tabela 32 – Comparativo global dos resultados das PWA's*

| | PWA- Ins. Chrome | PWA Chrome | PWA - Ins. Firefox | PWA - FireFox | PWA Instalada Opera | PWA - Opera |
|---|---|---|---|---|---|---|
| **Login** | 1606,6 | 1708,8 | 2074,8 | 1877 | 1849,2 | 1671,2 |
| **Registar** | 1024,4 | 999,6 | 1154 | 1012,8 | 1122 | 1016,2 |
| **Imagem do disco** | 100,4 | 102,2 | 347,2 | 309,2 | 97 | 105,6 |
| **Imagem da rede** | 39 | 35,8 | 196,4 | 122,6 | 40,4 | 34,4 |
| **CPU ST** | 2816 | 2800,4 | 3961 | 3971,6 | 2810,6 | 2811,6 |
| **CPU MT** | 15858,2 | 15805 | 18953,2 | 19002,4 | 15691,2 | 15558,6 |
| **Pedidos de rede** | 13208,8 | 12224,2 | 12224,2 | 11039,2 | 12435 | 11833,4 |

Com esta primeira análise global dos resultados podemos verificar que o browser em que a PWA corre tem efectivamente interferência significativa nos resultados, denotando-se em especifico os testes referentes ao Firefox, tendo obtido o mesmo sem margem de duvida os piores resultados.

Com esta conclusão procedeu-se então a uma nova análise dos resultados desta vez apenas dos dois restantes browsers, Chrome e Opera, identificando os melhores e piores resultados como demonstrado na Tabela 33.

*Tabela 33 – Comparativo de resultados PWA nos browsers Chrome e Opera*

| | PWA- Ins. Chrome | PWA Chrome | PWA Instalada Opera | PWA - Opera |
|---|---|---|---|---|
| **Login** | 1606,6 | 1708,8 | 1849,2 | 1671,2 |
| **Registar** | 1024,4 | 999,6 | 1122 | 1016,2 |
| **Imagem do disco** | 100,4 | 102,2 | 97 | 105,6 |
| **Imagem da rede** | 39 | 35,8 | 40,4 | 34,4 |
| **CPU ST** | 2816 | 2800,4 | 2810,6 | 2811,6 |
| **CPU MT** | 15858,2 | 15805 | 15691,2 | 15558,6 |
| **Pedidos de rede** | 13208,8 | 12224,2 | 12435 | 11833,4 |

Existindo uma certa dispersão dos melhores e priores resultados, notando, contudo, que as versões que correm directamente no browser embora de forma muito ligeira apresentam melhores resultados, foi elaborada uma nova tabela apenas com esses mesmos resultados e

procedeu-se novamente a uma identificação dos melhores e piores, como se pode verificar na Tabela 34.

*Tabela 34 – Dois melhores resultados dos testes das PWA*

| | PWA Chrome | PWA - Opera |
|---|---|---|
| Login | 1708,8 | 1671,2 |
| Registar | 999,6 | 1016,2 |
| Imagem do disco | 102,2 | 105,6 |
| Imagem da rede | 35,8 | 34,4 |
| CPU ST | 2800,4 | 2811,6 |
| CPU MT | 15805 | 15558,6 |
| Pedidos de rede | 12224,2 | 11833,4 |

Após esta filtragem e tendo uma maior concentração de melhores resultados foram utilizados para uma comparação final entre as diversas tecnologias/frameworks, identificando os melhores, piores e resultados intermédios de forma a identificar mais claramente os melhores desempenhos entre ambas como se pode constatar na Tabela 35.

*Tabela 35 – Comparação final "filtrada" entre tecnologias/frameworks*

| | Android | PWA - Opera | IONIC |
|---|---|---|---|
| Login | 2264,8 | 1671,2 | 1583,4 |
| Registar | 1066,2 | 1016,2 | 1043,4 |
| Imagem do disco | 67,2 | 105,6 | 680,8 |
| Imagem da rede | 2940,4 | 34,4 | 256,2 |
| CPU ST | 2200,2 | 2811,6 | 3237,4 |
| CPU MT | 11638,4 | 15558,6 | 15884,6 |
| Pedidos de rede | 5088,4 | 11833,4 | 13190,6 |

Com uma breve análise e tendo em conta meramente a performance podemos obter um "pódio" sendo Android nativo o detentor dos melhores resultados de performance, embora tenha também alguns dos piores resultados, sendo IONIC a detentora dos piores resultados. Podemos, no entanto, verificar que PWA resultados mais equilibrados, obtendo dois dos melhores resultados e alguns resultados bastante próximos dos melhores.

Com base em todos os resultados e análises foi então elaborada uma tabela no Excel para servir de auxilio à escolha da tecnologia/framework que mais se enquadra nos projectos a desenvolver. A referida tabela é composta por perguntas sobre o projecto a que o utilizador deve responder sim ou não caso o seu projecto realize ou não a tarefa mencionada na pergunta, sendo de seguida feita referencia a importância dessa tarefa no aspecto global do projecto.

Com base nessas duas respostas do utilizador, é extrapolada uma cotação para cada tecnologia, sendo que, a não realização da tarefa equivale a 0, a utilização e tarefa com importância mencionada irá equivaler a uma pontuação de 1 multiplicado pela respectiva posição inversa à posição da tabela comparativa dos testes, ou seja, no caso das 3 tecnologias/frameworks utilizadas, a que obteve melhores resultados ira ter uma multiplicação de 3 em caso se importância da tarefa, caso a tarefa não seja importante pontuaram todas 1.

No caso da complexidade do código e facilidade de manutenção será multiplicado o valor por 2 nos casos das duas melhores que obtiveram resultados semelhantes e 1 para Android por apresentar complexidade ligeiramente superior.

Finalmente para o caso do objectivo de desenvolvimento para multiplataforma, serão multiplicadas por um factor de 2 as tecnologias que o permitam e 0 as que não o permitem. Podendo verificar um exemplo na Tabela 36.

*Tabela 36 – Exemplo de sugestão de tecnologia/Framework a utilizar*

| | S/N | Funcionalidade mt Importante? | Android | PWA | IONIC |
|---|---|---|---|---|---|
| A aplicação efectua muitos pedidos isolados de rede como login ou registo? | Sim | Não | 1 | 1 | 1 |
| A aplicação carrega na sua maioria imagens do armazenamento local? | Sim | Não | 1 | 1 | 1 |
| A aplicação carrega na sua maioria imagens de endereçamento em servidor remoto? | Sim | Sim | 1 | 3 | 2 |
| A aplicação efectua muitas tarefas de processamento intensivo? | Sim | Não | 1 | 1 | 1 |
| A aplicação efectua muitos pedidos de rede interligados? | Sim | Não | 1 | 1 | 1 |
| Pretende desenvolver um único código para várias plataformas? | Sim | Sim | 0 | 2 | 2 |
| Simplicidade e facilidade de manutenção do código é importante? | Sim | Sim | 1 | 2 | 2 |
| Totais | | | 6 | 11 | 10 |

Por ultimo menciona-se que todo o código referente às aplicações desenvolvidas se encontra disponível para consulta e contribuição em https://github.com/ricardoSa84/mobileCompare e se irá tentar dar continuação a este projecto, melhorando o código existente bem como adicionando os testes em falta, bem como elaborar uma melhor ferramenta de sugestão de tecnologia/framework a utilizar.

# 9 Conclusões

Após a análise de todos os dados obtidos na realização desta dissertação, elaboração do código e posterior análise da simplicidade e facilidade de manutenção do código, pensamos ser necessário mencionar conclusões para dois cenários, um onde a performance é extremamente importante e não se tem em conta mais nenhum factor alem da performance da aplicação, e um outro em que se pretende uma solução equilibrada, tendo em conta todas as variáveis testadas.

No primeiro cenário de apenas dar importância à performance, excepto se a solução se basear em carregamento de imagens da rede, é recomendável a utilização de Android nativo, sendo que obteve os melhores resultados na quase totalidade dos testes efectuados, sendo que isto irá envolver maiores custos de desenvolvimento caso se tencione ter presença em diversas plataformas. Neste mesmo cenário também se irá verificar uma maior complexidade na manutenção do código, não só devido a maior complexidade como principalmente à também necessária manutenção a cada código de cada plataforma.

Num outro cenário em que se pretende uma solução não apenas baseada na performance, mas sim na obtenção de uma solução equilibrada, em que exista uma boa relação entre performance, facilidade de manutenção, e todos os factores envolvidos no desenvolvimento do projecto. Neste caso a solução recomendável de utilizar tendo em conta os diversos resultados obtidos nos testes será a elaboração de uma PWA. Embora esta apresente resultados um pouco diferentes consoante o browser utilizado, podemos mesmo assim verificar que seria a solução mais equilibrada, apresentando os melhores resultados intermédios, em certos casos obtendo mesmo melhores resultados que Android nativo. Para além de apresentar este tipo de resultados podemos também verificar que apresenta em certas situações uma maior simplicidade na realização de código, apenas necessitamos de um código para todo o tipo de plataformas, sendo na sua essência um site, as actualizações são imediatas, chegando aos utilizadores de forma instantânea, não sendo necessários updates como os verificados na stores das aplicações nativas, não é necessário o pagamento de cotas anuais no caso da Apple e única do caso do Android, mantendo no entanto a possibilidade de ser instalada nos dispositivos e executada de forma semelhante a uma aplicação nativa normal. Uma desvantagem que se poderá verificar será no caso de não existir necessidade

de nenhum alojamento online como presença web ou existência de algum género de API, sendo então necessária uma análise financeira aos custos de desenvolvimento das plataformas pretendidas, custos anuais das stores onde a aplicação irá ser publicada, bem como a soluções de alojamento, sendo que em certos casos se consegue pelo menos numa fase inicial alojamento online, sendo que se deverá sempre ter em conta uma análise aos custos de acordo com o crescimento expectável.

Tendo estes dois cenários em conta, e sendo que a probabilidade do segundo cenário ser muito mais usual, será então a utilização de uma PWA o cenário que se pensa ser mais recomendável devido aos factores já mencionados, permitindo obter na generalidade dos casos uma solução mais equilibrada e mais económica.

## 9.1 Limitações & trabalho futuro

Apesar de se ter efectuado uma série de testes a diversos casos, existiram algumas limitações, entre elas o tempo disponível para a realização desta dissertação, sendo que iram sempre existir mais testes possíveis de realizar e plataformas/frameworks para comparar. Apresentando então uma listagem de alguns pontos que se considera importantes de realizar numa continuação deste projecto que se pretende realizar.

- Elaboração de testes para iOS
- Elaboração de testes PWA no Browser Safari
- Elaboração de testes com acesso a funcionalidades do telemóvel como uso de GPS, acelerómetro, etc.
- Elaboração dos mesmos testes efectuados à PWA desenvolvida com recurso a outras framework como AngularJS, ReactJS, Laravel, Symfony, etc.
- Melhoramento do código das soluções existentes nomeadamente em relação a tratamento de erros
- Realização de outros tipos de testes que sejam significativos

# 10 Bibliografia

[1] Google, “Kotlinlang,” [Online]. Available: https://kotlinlang.org/docs/reference/comparison-to-java.html. [Acedido em 9 2018].

[2] “What is the Android application Architecture?,” [Online]. Available: https://www.quora.com/What-is-the-Android-application-Architecture. [Acedido em 12 11 2018].

[3] F. Ltd, “A Comparative Study: Objective-C Versus Swift,” [Online]. Available: https://medium.com/@fluperofficial/a-comparative-study-objective-c-versus-swift-9f6cebffed2. [Acedido em 1 2018].

[4] “Ionicframework, Angular, Cordova – Découvrir ionicframework pour créer une application iOS,” [Online]. Available: http://flaven.fr/2016/04/ionicframework-angular-cordova-decouvrir-ionicframework-pour-creer-une-application-ios/. [Acedido em 12 11 2018].

[5] “cordova,” [Online]. Available: https://cordova.apache.org/. [Acedido em 12 11 2018].

[6] I. Cruxlab, “Xamarin vs Ionic vs React Native: differences under the hood,” [Online]. Available: https://medium.com/swlh/xamarin-vs-ionic-vs-react-native-differences-under-the-hood-6b9cc3d2c826. [Acedido em 1 2018].

[7] “serviceWorkerImg,” [Online]. Available: https://www.keycdn.com/blog/progressive-web-apps. [Acedido em 12 11 2018].

[8] “Progressive Web Components,” [Online]. Available: https://www.innoq.com/en/blog/progressive-web-components-goto2016/#s18. [Acedido em 12 11 2018].

[9] Google, “Progressive Web Apps,” [Online]. Available: https://developers.google.com/web/progressive-web-apps/. [Acedido em 9 2018].

[10] "Progressive web apps, app shell," [Online]. Available: https://developers.google.com/web/ilt/pwa/introduction-to-progressive-web-app-architectures. [Acedido em 9 2018].

[11] A. Bar, "What Web Can Do Today," [Online]. Available: https://whatwebcando.today/. [Acedido em 1 2018].

[12] "Compatibilidade service worker," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorker. [Acedido em 9 2018].

[13] "State of JavaScript 2017," [Online]. Available: https://2017.stateofjs.com/2017/front-end/results/. [Acedido em 12 11 2018].

[14] S. Krause, "js-framework-benchmark," [Online]. Available: https://github.com/krausest/js-framework-benchmark. [Acedido em 1 2018].

[15] "Testes funcionais Vs. não funcionais," [Online]. Available: https://www.guru99.com/functional-testing-vs-non-functional-testing.html. [Acedido em 9 2018].

[16] "Static and dynamic testing in the software development life cycle," [Online]. Available: https://www.ibm.com/developerworks/library/se-static/index.html. [Acedido em 9 2018].

[17] K. K. Dipika Kelkar, "Analysis and Comparison of Performance Testing Tools," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* pp. 1880-1883, 2015.

[18] R. B. Khan, "Comparative Study of Performance," 2016.

[19] J. Colantonio, "Top 11 Open Source Performance Testing Tools for Load & Stress Testing," 18 Julho 2018. [Online]. Available: https://www.joecolantonio.com/open-source-performance-testing-tools/. [Acedido em 22 Outubro 2018].

[20] "Web performance testing: Top 12 free and open source tools to consider," [Online]. Available: https://techbeacon.com/web-performance-testing-top-12-free-open-source-tools-consider. [Acedido em 22 Outubro 2018].

[21] "Progressive web apps, lighthouse," [Online]. Available: https://developers.google.com/web/ilt/pwa/lab-offline-quickstart. [Acedido em 9 2018].